

UNCLASSIFIED

AL-ES1117  
Copy 17 of 42 copies

(2)

AD-A208 054

IDA DOCUMENT D-574

## AN Ada/SQL TUTORIAL

Bill R. Brykczynski  
Kerry Hilliard

February 1989

DTIC  
ELECTE  
MAY 22 1989  
S H D

Prepared for  
WIS Joint Program Management Office

**DISTRIBUTION STATEMENT A**

Approved for public release  
Distribution Unlimited

**IDA**

INSTITUTE FOR DEFENSE ANALYSES

1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

UNCLASSIFIED

IDA Log No. HQ 89-34163

## DEFINITIONS

IDA publishes the following documents to report the results of its work.

### Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, or (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

### Papers

Papers normally address relatively restricted technical or policy issues. They communicate the results of special analyses, interim reports or phases of a task, ad hoc or quick reaction work. Papers are reviewed to ensure that they meet standards similar to those expected of refereed papers in professional journals.

### Documents

IDA Documents are used for the convenience of the sponsors or the analysts to record substantive work done in quick reaction studies and major interactive technical support activities; to make available preliminary and tentative results of analyses or of working group and panel activities; to forward information that is essentially unanalyzed and unevaluated; or to make a record of conferences, meetings, or briefings, or of data developed in the course of an investigation. Review of Documents is suited to their content and intended use.

The results of IDA work are also conveyed by briefings and informal memoranda to sponsors and others designated by the sponsors, when appropriate.

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This IDA Document is published in order to make available the material it contains for the use and convenience of interested parties. The material has not necessarily been completely evaluated and analyzed, nor subjected to IDA review.

### DISCLAIMER OF WARRANTY AND LIABILITY

This is experimental prototype software. It is provided "as is" without warranty or representation of any kind. The Institute for Defense Analyses (IDA) does not warrant, guarantee, or make any representations regarding this software with respect to correctness, accuracy, reliability, merchantability, fitness for a particular purpose, or otherwise.

Users assume all risks in using this software. Neither IDA nor anyone else involved in the creation, production, or distribution of this software shall be liable for any damage, injury, or loss resulting from its use, whether such damage, injury, or loss is characterized as direct, indirect, consequential, incidental, special, or otherwise.

Approved for public release: distribution unlimited.

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified			1b RESTRICTIVE MARKINGS	
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release, unlimited distribution.	
2b DECLASSIFICATION/DOWNGRADING SCHEDULE				
4 PERFORMING ORGANIZATION REPORT NUMBER(S) IDA Document D-574			5 MONITORING ORGANIZATION REPORT NUMBER(S)	
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses		6b OFFICE SYMBOL IDA	7a NAME OF MONITORING ORGANIZATION OUSDA, DIMO	
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311			7b ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	
8a NAME OF FUNDING/SPONSORING ORGANIZATION WIS Joint Program Management Office		8b OFFICE SYMBOL (if applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031	
8c ADDRESS (City, State, and Zip Code) Room 5B19, The Pentagon Washington, D.C. 20330-6600			10 SOURCE OF FUNDING NUMBERS	
			PROGRAM ELEMENT NO.	PROJECT NO. T-W5-206
11 TITLE (Include Security Classification) An Ada/SQL Tutorial (U)				
12 PERSONAL AUTHOR(S) Bill R. Brykczynski, Kerry Hilliard				
13a TYPE OF REPORT Final	13b TIME COVERED FROM _____ TO _____		14 DATE OF REPORT (Year, Month, Day) 1989 February	15 PAGE COUNT 554
16 SUPPLEMENTARY NOTATION				
17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Ada, Structured Query Language (SQL), Ada/SQL, Database Management System (DBMS), language interface.	
FIELD	GROUP	SUB-GROUP		
19 ABSTRACT (Continue on reverse if necessary and identify by block number)  The purpose of IDA Document D-574. <i>An Ada/SQL Tutorial</i> , is to provide a beginner's guide to writing application programs using the Ada/Structured Query Language (SQL) Ada-Database Management System (DBMS) interface. General concepts of database management systems are provided, followed by a sequence of Structured Query Languages queries and the Ada/SQL counterparts.				
20 DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL Mr. Bill R. Brykczynski			22b TELEPHONE (Include area code) (703) 824-5515	22c OFFICE SYMBOL IDA/CSED

IDA DOCUMENT D-574

## AN Ada/SQL TUTORIAL

Bill R. Brykczynski  
Kerry Hilliard

February 1989



INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031  
Task T-W5-206



# UNCLASSIFIED

## CONTENTS

1. Introduction . . . . .	1
1.1 What Is A Database? . . . . .	1
1.2 What Is A Relational Database? . . . . .	2
1.3 What Is A Database Management System? . . . . .	3
1.4 What Is Structured Query Language? . . . . .	4
1.5 What Is An Application Program? . . . . .	5
1.6 What Is A Host Language Interface? . . . . .	5
1.7 What Is Ada/SQL? . . . . .	5
2. Hints For Setting Up A Database Through Your DBMS . . . . .	7
2.1 Table And Column Names . . . . .	7
2.2 Data Types . . . . .	7
2.3 Relationships Between Tables . . . . .	7
3. Introduction To Our Sample Database . . . . .	9
3.1 Organizing Our Database Into Tables . . . . .	9
3.2 Creating The Sample Database . . . . .	11
4. Interactive Queries . . . . .	17
4.1 Select & From . . . . .	17
4.2 INSERT INTO, The Basics . . . . .	19
4.3 More Basic SELECTs . . . . .	39
4.4 DISTINCT . . . . .	40
4.5 WHERE . . . . .	42
4.6 WHERE With Comparison Operators = <> < <= > >= . . . . .	42
4.7 WHERE With AND & OR . . . . .	45
4.8 WHERE With BETWEEN Operator . . . . .	47
4.9 WHERE With The IN Operator . . . . .	49
4.10 Wild Characters . . . . .	50
4.11 WHERE With LIKE Operator . . . . .	50
4.12 WHERE With The NOT Operator . . . . .	52
4.13 The Arithmetic Expressions + - * / . . . . .	55
4.14 The Aggregate Functions COUNT, MIN, MAX, SUM, AVG . . . . .	57
4.15 ORDER BY . . . . .	61
4.16 GROUP BY . . . . .	64
4.17 Nested Queries . . . . .	67
4.18 HAVING . . . . .	75
4.19 Joining Multiple Tables . . . . .	78
4.20 Correlation Names . . . . .	83
4.21 Self Joins . . . . .	85
4.22 EXISTS . . . . .	85
4.23 INSERT INTO . . . . .	89
4.24 UPDATE . . . . .	92
4.25 DELETE . . . . .	96
5. Introduction To Programming With Ada/SQL . . . . .	101
5.1 The Ada/SQL Library . . . . .	101
5.2 The Standard Modules . . . . .	101
5.3 Your Sublibrary . . . . .	101
5.4 The Application Scanner . . . . .	101

# UNCLASSIFIED

5.5	Compiling Ada/SQL Programs . . . . .	103
5.6	Linking Ada/SQL Programs . . . . .	104
5.7	Running Ada/SQL Programs . . . . .	104
6.	The Units Making Up An Ada/SQL Program . . . . .	105
6.1	Elements Permitted In The DDL . . . . .	105
6.2	Database Predefined Package . . . . .	106
6.3	Authorization Package' . . . . .	106
6.4	Data Type Definition Package . . . . .	107
6.5	Table Definition Package . . . . .	108
6.6	Variable Definition Package . . . . .	109
6.7	Package Body With DML Statements . . . . .	111
7.	The Ada Code For The DDL Units . . . . .	113
7.1	The Authorization Package . . . . .	113
7.2	The Data Type Definition Package . . . . .	113
7.3	The Table Definition Package . . . . .	117
7.4	The Variable Definition Package . . . . .	121
8.	The Basics For The Ada/SQL Program . . . . .	123
8.1	Conversion Subroutines . . . . .	123
8.2	Exceptions Likely To Be Encountered . . . . .	127
8.3	Skeleton Of The DML Unit . . . . .	127
9.	Getting Ready For Ada/SQL DML Queries . . . . .	133
9.1	OPEN_DATABASE . . . . .	133
9.2	EXIT_DATABASE . . . . .	134
9.3	Cursors . . . . .	134
10.	Ada/SQL DML Queries . . . . .	135
10.1	SELECT & FROM & FETCH & INTO . . . . .	136
10.2	INSERT INTO, The Basics . . . . .	143
10.3	More Basic SELECTS . . . . .	183
10.4	DISTINCT . . . . .	186
10.5	WHERE . . . . .	190
10.6	WHERE With Comparison Operators = <> < <= > >= . . . . .	191
10.7	WHERE With AND & OR . . . . .	204
10.8	WHERE With BETWEEN Operator . . . . .	212
10.9	WHERE With The IN Operator . . . . .	215
10.10	Wild Characters . . . . .	219
10.11	WHERE With LIKE Operator . . . . .	219
10.12	WHERE With The NOT Operator . . . . .	224
10.13	The Arithmetic Expressions + - * / . . . . .	233
10.14	The Aggregate Functions COUNT, MIN, MAX, SUM, AVG . . . . .	239
10.15	ORDER BY . . . . .	247
10.16	GROUP BY . . . . .	256
10.17	Nested Queries . . . . .	263
10.18	HAVING . . . . .	278
10.19	Joining Multiple Tables . . . . .	289
10.20	Correlation Names . . . . .	303
10.21	Self Joins . . . . .	307
10.22	EXISTS . . . . .	310
10.23	INSERT INTO . . . . .	315

# UNCLASSIFIED

10.24	UPDATE . . . . .	325
10.25	DELETE_FROM . . . . .	336
11.	All The Pieces Of The Sample Program . . . . .	352
11.1	The Authorization Package - AUTH_PACK.ADA . . . . .	352
11.2	The Data Type Definition Package - TYPES.ADA . . . . .	352
11.3	The Table Definition Package - TABLES.ADA . . . . .	353
11.4	The Variable Definition Package - VARIABLES.ADA . . . . .	354
11.5	The Conversion Package - CONVERSIONS.ADA . . . . .	356
11.6	The Sample Program - EXAMPLES.ADA . . . . .	360
11.7	Output From The Sample Program . . . . .	506

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

2  
INSPECTED  
QUALITY

UNCLASSIFIED

## PREFACE

The purpose of IDA Document D-574, *An Ada/SQL Tutorial*, is to provide a beginner's guide to writing application programs using the Ada/SQL Ada-DBMS interface. General concepts of database management systems are provided, followed by a sequence of Structured Query Languages (SQL) queries and the Ada/SQL counterparts.

The importance of this document is based on partial fulfillment of Task Order T-W5-206, WIS Application Software Study, which is to provide an Ada/SQL tutorial. As a Document, D-574 is directed toward users who are concerned with the development and operation of an Ada/SQL application.

## 1. Introduction

This report is a beginner's guide to writing application programs using the Ada/SQL system. This section of the document will begin by explaining the purpose of a database, the purpose of a database management system, what makes a database relational, what Structured Query Language is, and what the Ada/SQL system is. A sequence of examples will illustrate the creation of a database, the writing of an application program, the debugging of the program and ultimately the successful execution of the program. All examples will be initially shown as interactive SQL queries. Later, the same examples will be coded in an Ada/SQL program. The SQL is shown first so that the easy translation to Ada/SQL can be readily seen.

This report assumes that the reader is a programmer familiar with Ada applications. Examples shown in the report have been written using a DEC VAX/VMS<sup>1</sup> computer system with the Oracle<sup>2</sup> Relational Data Base Management System. The use of a different computer or database system may generate results similar, but not identical, to those which are illustrated in this report. The Ada/SQL application program will be identical to the examples used in this report, since Ada/SQL is standard from system to system. Interactive examples that are system dependent will be identified as they occur. It is not essential that the reader be familiar with a database system for this guide to be useful, however, it may add some additional understanding.

### 1.1 What Is A Database?

A database is an organized collection of information which serves some specific purpose. It is the storage of information that could be useful in the form of a list or table. A database contains entries which are words, numbers, dates or other pieces of information. Examples of data which could be stored as databases are mailing lists, customer lists, supplier lists, inventories, accounts, a telephone book or an encyclopedia. Anything that could be stored as a list or collection of information can also be stored as a database.

In order to use this information one would have to organize it in some form which would enable particular items of interest to be isolated. Take a telephone book for example. It contains a list of names, addresses and phone numbers. When using a telephone book, the item you are looking for is a phone number. Imagine a telephone book organized by phone numbers. It would be extremely difficult to find a particular phone number given such an organization. The list of numbers would have to be sequentially read in numerical order checking the names, which would not be in alphabetic order, for the desired number. How about organizing the list by the date which phone service was started. That would not help very much either. When we want to know someone's phone number we generally know their name, therefore to make a list most useful we organize our telephone books by name. Assume information is desired concerning a particular province in China. In looking up China in the encyclopedia, additional information would direct us to further geographical division, which in turn would direct us to information about the rivers in this province and the crops grown here etc. Here we see one referenced item pointing to another. Database information can be chained together in much the same way.

---

1. DEC, VAX, and VMS are registered trademarks of Digital Equipment Corporation.

2. Oracle is a registered trademark of Oracle Corporation.

## UNCLASSIFIED

Information in a database needs to be organized so that one can find the necessary data by starting with only a small piece of information, for example the name in the telephone book or China in the encyclopedia. This small piece of reference information is called a key. If a database has no keys you will have to read from the beginning until you find the information you're looking for, like the telephone book organized by phone number. This is why databases are organized around one or more keys making it possible to look things up in several different ways.

Information in a database is also organized with more general information pointing to or referencing more detailed information. The example of an encyclopedia entry of China would point to detailed information, like provinces, which in turn points to even more detailed information like rivers and crops grown. Frequently the detailed information will also have a pointer or reference back to the general information. For example, crop information would be detailed information about China. One of the entries on crops is rice, and it would point to China where it is a major crop.

Detailed information may be referenced by more than one general item. For example rice was referenced from the general entry on China. There is also a general entry on food crops, which points to grains, which points to the entry on rice, which points to China as a major grower. Detailed information for one entry may also be general information in another category. For example, the entry on China will point to the entry on rice through a chain of detail information. In addition, the general information about rice points to China through a chain of detail information.

The less information stored in a database the quicker it is to find any one particular item. Therefore, one should always try to avoid entering duplicate information in a database. To do this one must enter as a general item, any item of information which is general enough to be referenced on its own or as detailed information to another item. Then addition pointers should be added from this item to any more general items of interest. Likewise this item will be pointed to by the more general items.

Another concern when maintaining data in a database is keeping it current. Take the entry on rice, imagine that the general categories contained all the information about rice instead of a pointer to another entry on rice. The entry on China would contain a paragraph on rice and the entry on food crops would contain a paragraph on rice and the entry on grains would contain that same paragraph on rice. If a new strain of rice has been found, one must update all of information about rice. This would require changing the same information in three different places. In addition, one might be missing another place that contained a paragraph on rice. But if rice was an entry on its own, then the information could be changed in one place and it would be immediately available to all articles pointing to information on rice.

In conclusion, a database is nothing more than a list of information organized in such a way as to make data easy to find, non-redundant and with items referencing other items.

### 1.2 What Is A Relational Database?

There are many ways which the data in a database can be stored. A relational database refers to the organization and storing of data in a tabular structure with the ability to retrieve data in any order. Visualize a table with categories across the top and with individual cases listed down the left side. Let's use the telephone book example here. The categories would be name, address and phone number. The key to the individual cases is listed in the left most column of data, this is, of course, the name. To select a specific case from a table many types of databases would have to start at the beginning of the list and read each key in order to find the desired one. A relational database keeps track of these keys so it can select the desired one without reading all previous entries. Much the same way you'd flip back and forth in the telephone book before finding the correct entry.

## UNCLASSIFIED

A relational database contains a group of related tables. For example we could have three tables containing information about people. One would be a phone list and, like the telephone book, would contain a name, address and phone number. Another could be a birthday list and contain a name, birthday and favorite type of gift. Yet another could be a list of employers which would contain a name, employer's name, address, and number of years worked here. You can see that the relationship between these tables is a person's name. Why not simply put all this information into the same table? Because if you used some of the information frequently, such as the phone number, it would be a waste of time to sort through birthdays etc. to locate a phone number. Good relational database design calls for putting the least practical amount of data into each table and relating the tables to one another with a key, such as name here.

A table in a relational database may also be called a file or a relation. We will call them tables. Each entry in the table contains all the information descriptive of the key for that entry. A name, address and phone number make up one complete entry in the table. In a relational database an entry is called a record or a row. We will call them records. Each record has certain elements in it, such as name, address and phone number. These elements are the smallest items which you can address in the database. These elements are called columns, fields or attributes. We will call them columns.

Each column stores the same kind of information for each record. The name column in our phone book will always contain a name and never an address. Each record has the same number of columns. Every record in our phone book will contain data in the name, address and phone number column. If someone desired his address to be omitted from the list we could fill that column with spaces, but that column would still exist. Each record is unique. No two records will contain exactly the same data in all columns. The order of records or columns is not important.

In a relational database we do not have to store the individual records in any specific order, they generally get stored in the order in which they are entered into the database. Even the keys do not need to be in any order. In our phone book example the names do not have to be stored in alphabetic order for the database to quickly locate the desired record. Internal workings of the database keep track of our keys. Likewise we could store our columns in the order of number, address and then name and still be able to reference individual records by name. The key does not have to be the left most column. In a relational database any column can be a key and more than one column can be keys. Take our 911 emergency service for example, when a call comes in they automatically know the phone number from where the call is being placed. They then use the number as a key in the same database table we've been talking about to locate the address of the caller.

### 1.3 What Is A Database Management System?

A database management system, DBMS, is a tool for organizing, storing, maintaining, calculating, combining and retrieving information from a database. A relational database management system, RDBMS, is a database management system used with a relational database. When this report refers to DBMS from here on the term is used interchangeably with RDBMS.

Without a DBMS you would have no organized method for manipulation of the data within the database. Your tables would truly be just files and if you wanted information from them you would have to write a program to read through the file until the desired information was found. A DBMS will have simple predefined methods for using the stored data. An DBMS will also have methods for defining the tables, columns, keys and relationships between tables.

There are generally two ways to manipulate data through a DBMS. One is interactively using a very

## UNCLASSIFIED

simple english like language which might have a command like:

```
CREATE A TABLE TELFPHONE_BOOK WITH  
COLUMNS NAME ADDRESS NUMBER
```

or:

```
SELECT FROM TABLE TELEPHONE_BOOK  
THE RECORD WITH COLUMN NAME = 'JONES'
```

The other way is through the use of calls to the DBMS from within a program, where you might use a command like:

```
CREATE_TABLE (TELEPHONE_BOOK, NAME, ADDRESS, NUMBER)
```

to place a call to the CREATE\_TABLE routine of the DBMS telling it first the table name followed by the columns which it will contain. A command like:

```
SELECT (TELEPHONE_BOOK, 'JONES', '', '')
```

might be used to request that the DBMS read from the table TELEPHONE\_BOOK and return a record who's first column is "JONES" and the next two columns contain something unknown.

A DBMS will require that when you create a table you tell it what type of data is permitted in each column. For example name in our telephone book would be alphabetic data, address would be alphanumeric data and number would be numeric data. Different DBMSs will have different data types which they recognize. See a guide to your DBMS for more information on data types.

A DBMS may also allow you to place restrictions on the data permitted in the columns of a table. One such restriction is UNIQUE, where the data in a column must not be the same as in any other record in that table. Phone number would be unique in our example since no two people should have the same number. With the unique flag set in the DBMS if you attempted to add a record with a duplicate phone number to the table, the DBMS would not allow that record to be added. Another common restriction is NOT\_NULL where a column of a table would be required to contain data. Perhaps we would like our name column to be set as not\_null. Many other restrictions are possible, such as, a column containing a date could be set such that records could not be assigned a future date. Or a column containing a number could be restricted to contain a number between 1 and 100. Different DBMSs will have different restrictions which may be placed on your data. See a guide to your DBMS for more information on restrictions.

### 1.4 What Is Structured Query Language?

A query language is an interactive simple english like language used to manipulate information quickly in a database without having to write a program. You simply sit down at a terminal and enter your queries, no programming is necessary and people who know nothing about programming can still access the data. The examples we discussed above,

```
CREATE A TABLE TELEPHONE_BOOK WITH  
COLUMNS NAME ADDRESS NUMBER
```



and:

```
SELECT FROM TABLE TELEPHONE_BOOK
THE RECORD WITH COLUMN NAME = ''JONES''
```

are examples of a query language.

With so many DBMSs on the market if each had it's own query language it would become very confusing for the user to change from one DBMS to another. The American National Standards Institute has solved that problem by issuing standards for a query language called Structured Query Language (SQL either as an acronym or pronounced see-quill). Many relational DBMSs now adhere to this standard. The examples of a query language above are not ANSI SQL.

### 1.5 What Is An Application Program?

An Application program is a set of procedures designed to accomplish a task, such as keeping track of sales, orders, managing inventories, etc. With our telephone book example we would need application programs to insert new records into the table when new people were connected for phone service, to delete records from the table when service was disconnected, to update records in the table when people changed their address, and a program to print out a new list of phone numbers in a specific order.

### 1.6 What Is A Host Language Interface?

A host language interface is a group of program calls to the DBMS which can be coded in programs to manipulate data in the same way that SQL would be used interactively. Most DBMSs have host language interfaces, but they virtually always differ from one DBMS to the next. And they rarely have any relationship to the DBMS query language. No attempt at standardization of the interfaces has yet been attempted by DBMS manufactures.

### 1.7 What Is Ada/SQL?

Ada/SQL is the standard interface between the Ada programming language and any ANSI SQL DBMS. It allows programmers to define and manipulate the data in a database through an Ada application program using only standard, compilable Ada. The Ada/SQL interface is written to look very much like standard SQL making it easy to understand and to program. Ada/SQL is designed as an extension to ANSI SQL, adding Ada's type declaration and checking capabilities to SQL. Ada/SQL can be implemented with any database management system implementing Level 2 of the ANSI standard SQL. It allows programmers to design applications which combine the best features of Ada and the best features of SQL, making these applications far more powerful and flexible than applications based on either Ada or SQL alone.

Regardless of the DBMS the same Ada/SQL statements are used. Therefore the same application programs can be used with different DBMSs without needing to be changed. With Ada/SQL you have more flexibility in defining integrity constraints for the data than you do in a usual database management system. Ada/SQL provides portability of database definitions and application programs across computer systems as well as DBMSs.

The logical data structures of the underlying database are defined to the Ada application program with

**UNCLASSIFIED**

the schema definition language (Ada/SQL-DDL). The basic operations for manipulation (access, insertion, deletion) of data within the database are defined through the data manipulation language (Ada/SQL-DML).

## 2. Hints For Setting Up A Database Through Your DBMS

### 2.1 Table And Column Names

For databases that will be used with Ada/SQL table names and column names should be limited to 18 characters. Table name may be upper case or lower case but all table names must be the same case. Ada/SQL, like Ada, does not differentiate between cases. However since some DBMSs do have case differences Ada/SQL provides the capability for the user to specify all table and/or column names to be either upper or lower case. All table names in a database must be of the same case. All column names in a database must be of the same case. But table names and column names do not need to be the same case. Make table and column names descriptive, use `TELEPHONE_BOOK` instead of `TABLE1` etc. Table names must not be duplicated within a database and column names must not be duplicated within a table. Begin table and column names with alphabetic characters, embed no spaces in the names and do not use Ada/SQL reserved words.

### 2.2 Data Types

There are four data types, character strings, integers, floating point numbers and enumeration types, which may be used for column data in databases which will be used with Ada/SQL. Many DBMSs offer additional data types, such as date or money. Refrain from using a data type that is not compatible with Ada/SQL. A character string consists of a sequence of one or more characters of the ASCII character set, alphabetic, numeric or special characters. Integers are whole numbers without decimal places. Floating point numbers contain decimal notation and may be expressed with a decimal point or in scientific notation. Enumeration types are strictly Ada, but we have allowed for their use within databases. When defining enumeration types to a database use their integer equivalent. For example, if you wish to define the following type in Ada:

```
type ENUM_NUMBERS is ( ZERO, ONE, TWO, THREE );
```

you would define the column to the DBMS as integer with values of 0, 1, 2, 3. Likewise to define the Ada type:

```
type COLORS is ( RED, BLUE, YELLOW, GREEN );
```

you would define the column to the DBMS as integer with values of 0, 1, 2, and 3 corresponding to RED, BLUE, YELLOW and GREEN.

### 2.3 Relationships Between Tables

We had discussed earlier how some items in a database will point to other items. Let's look again at the group of related tables of a phone list, a birthday list and an employer list, where the common or relating item is name. Name could be a rather lengthy field to include in each record and we could even have two or more people with the same name. So instead of storing the full name in each table we could add another table that would list the names and assign a unique number or code to each one. Then the other tables would use this number instead of the full name. This would make locating relationships quicker since in most cases the code would be shorter than the full name.



### 3. Introduction To Our Sample Database

Through examples with a sample database we will first learn how to manipulate data using an interactive query language. Then we will learn how to do the same manipulations using Ada/SQL. Our sample database will be for United University which is a very small school. We will be setting up a database to keep track of information about the professors, the students and the courses offered. We will first create the tables that we need, then fill them with data, then practice manipulating the data. Then we will set up some practical applications which administrative personnel at the school will be using.

United University is divided into five departments, History, Math, Science, Language and Art. We have five professors teaching at the school. Each professor is assigned to a department and teaches most of the courses in that department, though a professor may teach a course in another department. The information which we'll need to know about each professor is name, number of years they've taught here and their salary. Each course taught at the school falls under one of the departments listed above. We will need to know which professor teaches which course and how many semester hours are in each course. Then we have the students. All students live in the dorm. We'll need to know which dorm room each student lives in, what state the student is from, his major and how many years has he been at the school. Each student is taking one or more class. We need to know which class each student is taking, their first and their second semester grade and final class grade. We also have guidelines for salaries of the professors based on number of years with the school. And one other bit of information the school likes to keep track of is the average grade earned in each course.

#### 3.1 Organizing Our Database Into Tables

The first thing to do is to figure out how to divide our information into useful tables. We want to store the least amount of practical data in each table and allow the tables to relate to one another. We'll need to store information on departments, professors, students, courses available, classes being taken by the students, salary guidelines and average course grades. So we'll create seven tables.

Let's set up our first table to contain information about each department. The only information we know is the name of the department. We will probably want to relate things to the departments, like which department a professor teaches in or which department a course is in. So let's assign a number to each department. We'll call the table DEPARTMENT and it will contain the following columns of the indicated data types.

Next let's define a table with information about the teachers at our school. We'll need to know their names, both first and last, and let's assign a number to each teacher for later references. Let's include a reference to the department in which they teach. We also need to include the number of years they have taught here and their salary. We'll call this table PROFESSOR and it will contain the following columns of the indicated data types.

The next logical table to define is for the courses taught at United University. Again we'll assign a number to each course for reference. We need to include the name of the course and the number of semester hours in the course. Let's also include a reference to the department which this course falls under and a reference to the teacher who teaches it. We'll call this table COURSE and it will contain the following columns of the indicated data types.

UNCLASSIFIED

table DEPARTMENT		
column	data type	description
DEPT_ID	integer 1 digit	a unique number for each department, we'll use 1, 2, 3 etc., and we have fewer than 10 departments so we can do with only one digit
DEPT_DESC	character string	the description for each department, 8 characters such as History or Math

table PROFESSOR		
column	data type	description
PROF_ID	integer 2 digits	a unique number for each professor, we will always have less then 100 professors so two digits will do
PROF_NAME	character string	the professor's last name 12 characters
PROF_FIRST	character string	the professor's first name 10 characters
PROF_DEPT	integer 1 digit	the number of the department, from the DEPARTMENT table, in which this professor teaches
PROF_YEARS	integer 2 digits	the number of years this professor has taught at United University
PROF_SALARY	floating point	this professor's salary 5 digits before 2 after decimal

Now we better set up a table for the students that are attending our school. We will assign a number to each student and include their name, first and last, their dorm room, home state, major and number of years at this school. This table will be called STUDENT and it will contain the following columns of the indicated data types.

Let's create a list of the classes that each student is taking. In this table we'll store references to the student, the course and the department and store information about the grades earned for each semester and a final grade. We'll put this information in a table called CLASS and it will contain the following columns of the indicated data types.

Now let's create the table with the salary guidelines for the professors. The information we will store here is for a given range of years what is the minimum and maximum suggested salary for a professor and based on the number of years of service what is the recommended annual raise.

# UNCLASSIFIED

table COURSE		
<i>column</i>	<i>data type</i>	<i>description</i>
COURSE_ID	integer 3 digits	a unique number for each course taught
COURSE_DEPT	integer 1 digit	the number of the department, from the DEPARTMENT table, which this course falls under
COURSE_DESC	character string 20 characters	description of the course, such as World History
COURSE_PROF	integer 2 digits	the id number of the professor, from the PROFESSOR table, who teaches this course
COURSE_HOURS	integer 1 digit	the number of semester hours for this course

There's one last thing to keep track of. It will be a list of the courses and the average overall grade earned by all students in that course. The only information we need to store here is a reference to the course and a grade. We'll put this in a table called GRADE and it will contain the following columns of the indicated data types.

## 3.2 Creating The Sample Database

Ada/SQL allows you to manipulate existing databases. Ada/SQL cannot create a database. So we must use the DBMS query language for table creation. After we have created the tables I will give you examples of the types of queries you might wish to have in an application program. Then I'll show you how to convert these SQL queries to Ada/SQL in an Ada program.

We are going to create the sample database with the seven tables discussed above. If you do not know how to logon to your DBMS in an interactive query mode this is the time to find out. I am using the Oracle RDBMS on a VAX/VMS machine. The query language used in the examples below is Oracle's which is SQL. Not all DBMSs conform exactly to SQL so you may have to modify my examples a little to work with your system. Your responses may vary somewhat from mine also. You may encounter errors which I am not considering here. If so check to see if the query is correct for your system and that all tables and column names are spelled correctly. Correct your query and try again.

Enter your DBMS and prepare to enter queries. The first thing we will do is create our tables. To do this we use the create table command. The basic format which we are using is:

```
create table TABLE_NAME
( COLUMN_1      column_type (size),
  .....      ..... ) ;
```

where column\_type is "char" for a column with a data type of character string and "number" for a column with a data type of integer or floating point. For a "char" column "size" is the maximum width of the field. For an integer "number" column "size" is the maximum number of digits, for example 3 to

UNCLASSIFIED

table STUDENT		
<i>column</i>	<i>data type</i>	<i>description</i>
ST_ID	integer 3 digits	a unique number for each student, we will always have less then 1000 students so three digits will do
ST_NAME	character string 12 characters	the student's last name
ST_FIRST	character string 10 characters	the student's first name
ST_ROOM	character string 4 characters	the dorm room number which the student lives in, all of our students live on campus, the first character of the room number will be an A, B or C depending on which building the room is in and the last three characters will be the room number, for example A135
ST_STATE	character string 2 characters	the abbreviation of the students home
ST_MAJOR	integer 1 digit	the number of the department, from the DEPARTMENT table, in which this student's major is in
ST_YEAR	integer 1 digits	the number of years this student has been at United University

be able to store up to 999. For a floating point "number" column "size" contains two numbers separated with a comma. The first number is the maximum total number of digits to be stored, including digits to both the left and right of the decimal. The second number is the maximum number of digits to the right of the decimal point. For example a size specification of "(7,2)" would allow five digits before the decimal and two after the decimal.

The create table command is not found in Ada/SQL. Your tables must be in existence before an Ada/SQL application program can manipulate data within them. Your DBMS command for creating tables may not be as shown above. However I have explained the basic format which I will be using to create our tables so you can follow along.

### Example 3.2.1

Create the DEPARTMENT table with the column DEPT\_ID of data type integer with a maximum of one digit and the column DEPT\_DESC of data type character string with a maximum width of eight characters.



UNCLASSIFIED

table CLASS		
<i>column</i>	<i>data type</i>	<i>description</i>
CLASS_STUDENT	integer 3 digits	the students id number, from the STUDENT table, of this student
CLASS_DEPT	integer 1 digit	the number of the department, from the DEPARTMENT table, which this class falls under
CLASS_COURSE	integer 3 digits	the course number, from the COURSE table, which describes this class
CLASS_SEM_1	floating point 3 digits before, 2 after	the grade, percentage, given to this student for the first semester of this class
CLASS_SEM_2	floating point 3 digits before, 2 after	the grade, percentage, given to this student for the second semester of this class
CLASS_GRADE	floating point 3 digits before, 2 after decimal	the grade, percentage, given to this student for this entire class

table SALARY		
<i>column</i>	<i>data type</i>	<i>description</i>
SAL_YEAR	integer 2 digits	the low end for the range of years employed at the school
SAL_END	integer 2 digits	the high end for the range of years employed at the school
SAL_MIN	floating point 5 digits before, 2 after decimal	the minimum salary recommended for employment in the above range of years
SAL_MAX	floating point 5 digits before, 2 after decimal	the maximum salary recommended for employment in the above range of years
SAL_RAISE	floating point 1 digits before, 3 after decimal	the annual percent of salary increase recommended for someone in this range

```

create table DEPARTMENT
( DEPT_ID      number (1),
  DEPT_DESC    char  (8) ) ;

```

UNCLASSIFIED

table GRADE		
<i>column</i>	<i>data type</i>	<i>description</i>
GRADE_COURSE	integer 3 digits	the course number, from the COURSE table, which describes this class
GRADE_AVERAGE	floating point 3 digits before, 2 after decimal	the average grade, percentage, given to the students in this course

**My DBMS responds:**

Table created.

If you're not using Oracle your responses may not be identical to mine. However they should be similar and should not indicate an error. From now on my response will follow my query. The semicolon terminates my query and indicates to the DBMS that I want it executed. Throughout the rest of this manual the response will immediately follow the query.

**Example 3.2.2**

Create the PROFESSOR table with the columns PROF\_ID of data type integer with a maximum of two digits, PROF\_NAME of data type character string with a maximum width of twelve characters, PROF\_FIRST of data type character string with a maximum width of ten characters, PROF\_DEPT of data type integer with a maximum of 1 digit, PROF\_YEARS of data type integer with a maximum width of 2 digits and PROF\_SALARY of data type floating point with a maximum of five digits before and two after the decimal.

```
create table PROFESSOR
( PROF_ID    number (2),
  PROF_NAME  char  (12),
  PROF_FIRST char  (10),
  PROF_DEPT  number (1),
  PROF_YEARS number (2),
  PROF_SALARY number (7,2) ) ;
```

Table created.

**Example 3.2.3**

Create the COURSE table with the columns COURSE\_ID of data type integer with a maximum of three digits, COURSE\_DEPT of data type integer with a maximum of one digit, COURSE\_DESC of data type character string with a maximum width of twenty characters, COURSE\_PROF of data type integer with a maximum of two digits and COURSE\_HOUES of data type integer with a maximum of one digit.

```
create table COURSE
( COURSE_ID    number (3),
  COURSE_DEPT  number (1),
  COURSE_DESC  char  (20),
  COURSE_PROF  number (2),
  COURSE_HOURS number (1) ) ;
```

UNCLASSIFIED

Table created.

**Example 3.2.4**

Create the STUDENT table with the columns ST\_ID of data type integer with a maximum of three digits, ST\_NAME of data type character string with a maximum width of twelve characters, ST\_FIRST of data type character string with a maximum width of ten characters, ST\_ROOM of data type character string with a maximum width of four characters, ST\_STATE of data type character string with a maximum width of two characters, ST\_MAJOR of data type integer with a maximum of one digit and ST\_YEAR of data type integer with a maximum of one digit.

```
create table STUDENT
( ST_ID      number (3),
  ST_NAME    char  (12),
  ST_FIRST   char  (10),
  ST_ROOM    char  (4),
  ST_STATE   char  (2),
  ST_MAJOR   number (1),
  ST_YEAR    number (1) ) ;
```

Table created.

**Example 3.2.5**

Create the CLASS table with the columns CLASS\_STUDENT of data type integer with a maximum of three digits, CLASS\_DEPT of data type integer with a maximum of one digit, CLASS\_COURSE of data type integer with a maximum of three digits, CLASS\_SEM\_1 of data type floating point with a maximum of five digits before and two after the decimal, CLASS\_SEM\_2 of data type floating point with a maximum of five digits before and two after the decimal, CLASS\_GRADE of data type floating point with a maximum of five digits before and two after the decimal.

```
create table CLASS
( CLASS_STUDENT number (3),
  CLASS_DEPT    number (1),
  CLASS_COURSE  number (3),
  CLASS_SEM_1   number (5,2),
  CLASS_SEM_2   number (5,2),
  CLASS_GRADE   number (5,2) ) ;
```

Table created.

**Example 3.2.6**

Create the SALARY table with the columns SAL\_YEAR of data type integer with a maximum of two digits, SAL\_END of data type integer with a maximum of two digits, SAL\_MIN of data type floating point with a maximum of five digits before and two after the decimal, SAL\_MAX of data type floating point with a maximum of five digits before and two after the decimal, SAL\_RAISE of data type floating point with a maximum of one digit before and three after the decimal.

```
create table SALARY
( SAL_YEAR number (2),
```

UNCLASSIFIED

```
SAL_END    number (2),  
SAL_MIN    number (7,2),  
SAL_MAX    number (7,2),  
SAL_RAISE  number (4,3) ) ;
```

**Example 3.2.7**

Create the GRADE table with the columns GRADE\_COURSE of data type integer with a maximum of three digits, GRADE\_AVERAGE of data type floating point with a maximum of three digits before and two after the decimal.

```
create table GRADE  
( GRADE_COURSE  number (3),  
  GRADE_AVERAGE number (5,2) ) ;
```

Table created.

If your DBMS did not complain about any of these queries you should now have the seven tables created.

## 4. Interactive Queries

I will now show you the interactive queries for which there are equivalent queries in Ada/SQL. First we will process all queries interactively and then through Ada/SQL.

### 4.1 Select & From

To retrieve information from one or more tables you will use a select clause which specifies the columns you wish to see and a from clause to indicate the tables from which to extract the column data. An asterisk (\*) may be used in place of column names to indicate all column names in the table. The columns will be displayed in the order stated in the select clause, if all columns are selected with the asterisk then the columns will be displayed in the order stated in the create table command. The format of the select clause is

```
select COLUMN_1, COLUMN_2, ... from TABLE ;
```

or

```
select * from TABLE ;
```

#### Example 4.1.1

To show all the records in the DEPARTMENT table enter the command:

```
select *
from DEPARTMENT ;
```

no records selected

The DBMS should tell you that the table currently has no records. We have created the tables but have not yet filled them with data.

#### Example 4.1.2

To show only one column of all the records in the DEPARTMENT table you could enter the command:

```
select DEPT_DESC
from DEPARTMENT ;
```

no records selected

Again the DBMS will tell you that the table is empty.

f the ASCII character set, alphabetic, numeric or special characters. Integers are whole numbers without decimal places. Floating point numbers contain decimal notation and may be expressed with a decimal point or in scientific notation. Enumeration types are strictly Ada, but we have allowed for their use within databases. When defining enumeration types to a database use their integer equivalent. For example, if you wish to define the following type in Ada:

```
type ENUM_NUMBERS is ( ZERO, ONE, TWO, THREE );
```

**UNCLASSIFIED**

you would define the column to the DBMS as integer with values of 0, 1, 2, 3. Likewise to define the Ada type:

```
type COLORS is ( RED, BLUE, YELLOW, GREEN );
```

you would define the column to the DBMS as integer with values of 0, 1, 2, and 3 corresponding to RED, BLUE, YELLOW and GREEN.

## 4.2 INSERT INTO, The Basics

The next step is to put data into the tables we've created. This is done with the "insert into" statement. In this section we will discuss only the most simple form of the "insert into" statement. More complex forms will be discussed in a later chapter. To add a record to a table you must specify the table name and information for each column. The format of the insert into statement is:

```
insert into TABLE
values
( COLUMN_1_DATA, COLUMN_2_DATA, ... ) ;
```

You must supply data for every column in the table. Character strings must be enclosed in single quotes. Character string columns must be the maximum full length of the column. When a character string field won't fill up the column it should be padded with spaces. The "empty" characters in a character string must be ascii spaces when using Ada/SQL. Some DBMSs will automatically pad with spaces. Others will pad with a null value. If you are not sure how your DBMS will pad fill it with spaces yourself.

### Example 4.2.1

Let's insert a record into the DEPARTMENT table. Ada/SQL requires that you insert one record at a time. Some DBMSs may allow you to insert several with one query, but we'll be doing it one at a time here to be compatible with Ada/SQL.

```
insert into DEPARTMENT
values
( 1, 'History ' ) ;
```

1 record created.

### Example 4.2.2

Now let's select all records and all fields from the DEPARTMENT table, using the same query we looked at earlier.

```
select *
from DEPARTMENT ;
```

```
DEPT_ID DEPT_DES
1 History
```

### Example 4.2.3

Now let's select only one field from all records from the DEPARTMENT table.

```
select DEPT_DESC
from DEPARTMENT ;
```

```
DEPT_DES
History
```

## UNCLASSIFIED

### Example 4.2.4

We'll finish filling up the DEPARTMENT table with all the records we plan to have in it.

```
insert into DEPARTMENT
values
( 2, 'Math ' ) ;
```

1 record created.

### Example 4.2.5

```
insert into DEPARTMENT
values
( 3, 'Science ' ) ;
```

1 record created.

### Example 4.2.6

```
insert into DEPARTMENT
values
( 4, 'Language' ) ;
```

1 record created.

### Example 4.2.7

```
insert into DEPARTMENT
values
( 5, 'Art ' ) ;
```

1 record created.

### Example 4.2.8

Let's display all the records which have been inserted into the DEPARTMENT table. Your list of records may not be ordered exactly as this example is. The ordering of records in a relational database is insignificant. Later on we will discuss how to list records in a specified order.

```
select *
from DEPARTMENT ;
```

```
DEPT_ID DEPT_DES
1 History
2 Math
3 Science
4 Language
5 Art
```



UNCLASSIFIED

**Example 4.2.9**

The next several examples will fill the PROFESSOR table.

```
insert into PROFESSOR
values
( 01, 'Dysart  ', 'Gregory ', 3, 03, 35000.00 ) ;
```

1 record created.

**Example 4.2.10**

```
insert into
PROFESSOR values
( 02, 'Hall    ', 'Elizabeth ', 4, 07, 45000.00 ) ;
```

1 record created.

**Example 4.2.11**

```
insert into PROFESSOR
values
( 03, 'Steinbacner ', 'Moris   ', 2, 01, 30000.00 ) ;
```

1 record created.

**Example 4.2.12**

```
insert into PROFESSOR
values
( 04, 'Bailey   ', 'Bruce   ', 5, 15, 50000.00 ) ;
```

1 record created.

**Example 4.2.13**

```
insert into PROFESSOR
values
( 05, 'Clements ', 'Carol   ', 1, 04, 40000.00 ) ;
```

1 record created.

**Example 4.2.14**

And now we'll take a look at the records in the PROFESSOR table.

```
select *
from PROFESSOR ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00

# UNCLASSIFIED

2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

## Example 4.2.15

We now fill the COURSE table with data.

```
insert into COURSE
values
( 101, 1, 'World History', 05, 2 ) ;
```

1 record created.

## Example 4.2.16

```
insert into COURSE
values
( 102, 1, 'Political History', 05, 3 ) ;
```

1 record created.

## Example 4.2.17

```
insert into COURSE
values
( 103, 1, 'Ancient History', 05, 2 ) ;
```

1 record created.

## Example 4.2.18

```
insert into COURSE
values
( 201, 2, 'Algebra', 03, 4 ) ;
```

1 record created.

## Example 4.2.19

```
insert into COURSE
values
( 202, 2, 'Geometry', 03, 4 ) ;
```

1 record created.

## Example 4.2.20

```
insert into COURSE
values
( 203, 2, 'Trigonometry', 03, 5 ) ;
```

UNCLASSIFIED

1 record created.

**Example 4.2.21**

```
insert into COURSE
values
( 204, 2, 'Calculus      ', 03, 4 ) ;
```

1 record created.

**Example 4.2.22**

```
insert into COURSE
values
( 301, 3, 'Chemistry     ', 01, 3 ) ;
```

1 record created.

**Example 4.2.23**

```
insert into COURSE
values
( 302, 3, 'Physics       ', 01, 5 ) ;
```

1 record created.

**Example 4.2.24**

```
insert into COURSE
values
( 303, 3, 'Biology       ', 01, 4 ) ;
```

1 record created.

**Example 4.2.25**

```
insert into COURSE
values
( 401, 4, 'French        ', 02, 2 ) ;
```

1 record created.

**Example 4.2.26**

```
insert into COURSE
values
( 402, 4, 'Spanish       ', 05, 2 ) ;
```

1 record created.

**Example 4.2.27**

UNCLASSIFIED

```
insert into COURSE
values
( 403, 4, 'Russian      ', 02, 4 ) ;
```

1 record created.

**Example 4.2.28**

```
insert into COURSE
values
( 501, 5, 'Sculpture    ', 04, 1 ) ;
```

1 record created.

**Example 4.2.29**

```
insert into COURSE
values
( 502, 5, 'Music        ', 04, 1 ) ;
```

1 record created.

**Example 4.2.30**

```
insert into COURSE
values
( 503, 5, 'Dance        ', 05, 2 ) ;
```

1 record created.

**Example 4.2.31**

List all the records currently in the COURSE table.

```
select *
from COURSE ;
```

COURSE_ID	COURSE_DEPT	COURSE_DESC	COURSE_PROF	COURSE_HOURS
101	1	World History	5	2
102	1	History	5	3
103	1	Ancient History	5	2
201	2	Algebra	3	4
202	2	Geometry	3	4
203	2	Trigonometry	3	5
204	2	Calculus	3	4
301	3	Chemistry	1	3
302	3	Physics	1	5
303	3	Biology	1	4
401	4	French	2	2
402	4	Spanish	5	2

# UNCLASSIFIED

403	4 Russian	2	4
501	5 Sculpture	4	1
502	5 Music	4	1
503	5 Dance	5	2

16 records selected.

## Example 4.2.32

And now fill up the STUDENT table with data.

```
insert into STUDENT
values
( 001, 'Horrigan ', 'William ', 'A101', 'VA', 3, 4 ) ;
```

1 record created.

## Example 4.2.33

```
insert into STUDENT
values
( 002, 'McGinn ', 'Gregory ', 'A102', 'MD', 1, 3 ) ;
```

1 record created.

## Example 4.2.34

```
insert into STUDENT
values
( 003, 'Lewis ', 'Molly ', 'A103', 'PA', 4, 2 ) ;
```

1 record created.

## Example 4.2.35

```
insert into STUDENT
values
( 004, 'Waxler ', 'Dennis ', 'A104', 'NC', 2, 2 ) ;
```

1 record created.

## Example 4.2.36

```
insert into STUDENT
values
( 005, 'McNamara ', 'Howard ', 'A201', 'VA', 5, 1 ) ;
```

1 record created.

## Example 4.2.37

UNCLASSIFIED

```
insert into STUDENT
values
( 006, 'Hess      ', 'Fay      ', 'A202', 'DC', 3, 3 ) ;
```

1 record created.

**Example 4.2.38**

```
insert into STUDENT
values
( 007, 'Guiffre   ', 'Jennifer ', 'A203', 'MD', 4, 1 ) ;
```

1 record created.

**Example 4.2.39**

```
insert into STUDENT
values
( 008, 'Hagan     ', 'Carl     ', 'A204', 'PA', 5, 4 ) ;
```

1 record created.

**Example 4.2.40**

```
insert into STUDENT
values
( 009, 'Bearman   ', 'Rose     ', 'A301', 'VA', 2, 1 ) ;
```

1 record created.

**Example 4.2.41**

```
insert into STUDENT
values
( 010, 'Thompson  ', 'Paul     ', 'A302', 'NC', 1, 3 ) ;
```

1 record created.

**Example 4.2.42**

```
insert into STUDENT
values
( 011, 'Bennett   ', 'Nellie   ', 'A303', 'PA', 4, 3 ) ;
```

1 record created.

**Example 4.2.43**

```
insert into STUDENT
values
( 012, 'Schmidt   ', 'John     ', 'A304', 'SC', 5, 2 ) ;
```

UNCLASSIFIED

1 record created.

**Example 4.2.44**

```
insert into STUDENT
values
( 013, 'Gevarter ', 'Susan ', 'B101', 'NY', 5, 4 ) ;
```

1 record created.

**Example 4.2.45**

```
insert into STUDENT
values
( 014, 'Sherman ', 'Donald ', 'B102', 'VA', 3, 3 ) ;
```

1 record created.

**Example 4.2.46**

```
insert into STUDENT
values
( 015, 'Gorham ', 'Milton ', 'B103', 'WV', 2, 2 ) ;
```

1 record created.

**Example 4.2.47**

```
insert into STUDENT
values
( 016, 'Williams ', 'Alvin ', 'B104', 'DC', 1, 1 ) ;
```

1 record created.

**Example 4.2.48**

```
insert into STUDENT
values
( 017, 'Woodliff ', 'Dorothy ', 'B201', 'MD', 4, 4 ) ;
```

1 record created.

**Example 4.2.49**

```
insert into STUDENT
values
( 018, 'Ratliff ', 'Ann ', 'B202', 'NY', 5, 1 ) ;
```

1 record created.

**Example 4.2.50**

UNCLASSIFIED

```
insert into STUDENT
values
( 019, 'Phung      ', 'Kim      ', 'B203', 'SC', 2, 2 ) ;
```

1 record created.

**Example 4.2.51**

```
insert into STUDENT
values
( 020, 'McMurray  ', 'Eric     ', 'B204', 'VA', 2, 1 ) ;
```

1 record created.

**Example 4.2.52**

```
insert into STUDENT
values
( 021, 'O''Leary   ', 'Peggy    ', 'C101', 'PA', 3, 4 ) ;
```

1 record created.

**Example 4.2.53**

```
insert into STUDENT
values
( 022, 'Martin    ', 'Charlotte ', 'C102', 'DC', 1, 2 ) ;
```

1 record created.

**Example 4.2.54**

```
insert into STUDENT
values
( 023, 'O''Day     ', 'Hilda    ', 'C103', 'NC', 4, 1 ) ;
```

1 record created.

**Example 4.2.55**

```
insert into STUDENT
values
( 024, 'Martin    ', 'Edward   ', 'C104', 'MD', 5, 3 ) ;
```

1 record created.

**Example 4.2.56**

```
insert into STUDENT
values
( 025, 'Chateauneuf ', 'Chelsea  ', 'C105', 'VA', 1, 3 ) ;
```



# UNCLASSIFIED

1 record created.

## Example 4.2.57

List all the records stored in the STUDENT table.

```
select *
from STUDENT ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	4
2	McGinn	Gregory	A102	MD	1	3
3	Lewis	Molly	A103	PA	4	2
4	Waxler	Dennis	A104	NC	2	2
5	McNamara	Howard	A201	VA	5	1
6	Hess	Fay	A202	DC	3	3
7	Guiffre	Jennifer	A203	MD	4	1
8	Hagan	Carl	A204	PA	5	4
9	Bearman	Rose	A301	VA	2	1
10	Thompson	Paul	A302	NC	1	3
11	Bennett	Nellie	A303	PA	4	3
12	Schmidt	John	A304	SC	5	2
13	Gevarter	Susan	B101	NY	5	4
14	Sherman	Donald	B102	VA	3	3
15	Gorham	Milton	B103	WV	2	2
16	Williams	Alvin	B104	DC	1	1
17	Woodliff	Dorothy	B201	MD	4	4
18	Ratliff	Ann	B202	N	5	1
19	Phung	Kim	B203	SC	2	2
20	McMurray	Eric	B204	VA	2	1
21	O'Leary	Peggy	C101	PA	3	4
22	Martin	Charoltte	C102	DC	1	2
23	O'Day	Hilda	C103	NC	4	1
24	Martin	Edward	C104	MD	5	3
25	Chateaufneuf	Chelsea	C105	VA	1	3

25 records selected.

## Example 4.2.58

Now fill up the CLASS table with information.

```
insert into CLASS
values
( 001, 3, 302, 089.49, 051.91, 000.00 ) ;
```

1 record created.

## Example 4.2.59

UNCLASSIFIED

```
insert into CLASS
values
( 001, 3, 303, 077.61, 088.84, 000.00 ) ;
```

1 record created.

**Example 4.2.60**

```
insert into CLASS
values
( 002, 1, 103, 054.38, 084.77, 000.00 ) ;
```

1 record created.

**Example 4.2.61**

```
insert into CLASS
values
( 003, 4, 403, 092.92, 097.48, 000.00 ) ;
```

1 record created.

**Example 4.2.62**

```
insert into CLASS
values
( 004, 2, 204, 071.17, 070.55, 000.00 ) ;
```

1 record created.

**Example 4.2.63**

```
insert into CLASS
values
( 005, 5, 503, 088.83, 081.12, 000.00 ) ;
```

1 record created.

**Example 4.2.64**

```
insert into CLASS
values
( 006, 3, 301, 066.26, 094.60, 000.00 ) ;
```

1 record created.

**Example 4.2.65**

```
insert into CLASS
values
( 006, 4, 402, 100.00, 100.00, 000.00 ) ;
```

UNCLASSIFIED

1 record created.

**Example 4.2.66**

```
insert into CLASS
values
( 007, 4, 401, 100.00, 100.00, 000.00 ) ;
```

1 record created.

**Example 4.2.67**

```
insert into CLASS
values
( 007, 4, 402, 100.00, 100.00, 000.00 ) ;
```

1 record created.

**Example 4.2.68**

```
insert into CLASS
values
( 007, 4, 403, 100.00, 100.00, 000.00 ) ;
```

1 record created.

**Example 4.2.69**

```
insert into CLASS
values
( 007, 5, 503, 100.00, 100.00, 000.00 ) ;
```

1 record created.

**Example 4.2.70**

```
insert into CLASS
values
( 008, 5, 502, 069.68, 056.92, 000.00 ) ;
```

1 record created.

**Example 4.2.71**

```
insert into CLASS
values
( 009, 2, 204, 055.53, 089.81, 000.00 ) ;
```

1 record created.

**Example 4.2.72**

UNCLASSIFIED

```
insert into CLASS
values
( 010, 1, 102, 093.72, 099.55, 000.00 ) ;
```

1 record created.

**Example 4.2.73**

```
insert into CLASS
values
( 011, 4, 401, 081.99, 076.29, 000.00 ) ;
```

1 record created.

**Example 4.2.74**

```
insert into CLASS
values
( 012, 5, 501, 075.81, 083.03, 000.00 ) ;
```

1 record created.

**Example 4.2.75**

```
insert into CLASS
values
( 013, 5, 502, 067.36, 080.15, 000.00 ) ;
```

1 record created.

**Example 4.2.76**

```
insert into CLASS
values
( 014, 3, 302, 092.27, 082.47, 000.00 ) ;
```

1 record created.

**Example 4.2.77**

```
insert into CLASS
values
( 015, 2, 202, 089.75, 095.74, 000.00 ) ;
```

1 record created.

**Example 4.2.78**

```
insert into CLASS
values
( 016, 1, 101, 085.64, 078.26, 000.00 ) ;
```

UNCLASSIFIED

1 record created.

**Example 4.2.79**

```
insert into CLASS
values
( 016, 1, 101, 094.59, 091.52, 000.00 ) ;
```

1 record created.

**Example 4.2.80**

```
insert into CLASS
values
( 016, 2, 204, 083.40, 094.88, 000.00 ) ;
```

1 record created.

**Example 4.2.81**

```
insert into CLASS
values
( 016, 3, 302, 082.14, 087.11, 000.00 ) ;
```

1 record created.

**Example 4.2.82**

```
insert into CLASS
values
( 016, 4, 403, 089.32, 097.40, 000.00 ) ;
```

1 record created.

**Example 4.2.83**

```
insert into CLASS
values
```

1 record created.

**Example 4.2.84**

```
insert into CLASS
values
( 017, 4, 401, 094.71, 063.36, 000.00 ) ;
```

1 record created.

**Example 4.2.85**

UNCLASSIFIED

```
insert into CLASS
values
( 018, 5, 503, 092.69, 071.69, 000.00 ) ;
```

1 record created.

**Example 4.2.86**

```
insert into CLASS
values
( 019, 2, 201, 081.31, 095.95, 000.00 ) ;
```

1 record created.

**Example 4.2.87**

```
insert into CLASS
values
( 020, 2, 204, 088.28, 079.01, 000.00 ) ;
```

1 record created.

**Example 4.2.88**

```
insert into CLASS
values
( 021, 3, 303, 071.16, 074.14, 000.00 ) ;
```

1 record created.

**Example 4.2.89**

```
insert into CLASS
values
( 022, 1, 102, 058.97, 086.58, 000.00 ) ;
```

1 record created.

**Example 4.2.90**

```
insert into CLASS
values
( 022, 2, 201, 081.75, 092.97, 000.00 ) ;
```

1 record created.

**Example 4.2.91**

```
insert into CLASS
values
( 022, 5, 503, 074.49, 098.30, 000.00 ) ;
```

# UNCLASSIFIED

1 record created.

## Example 4.2.92

```
insert into CLASS
values
( 023, 4, 402, 096.33, 081.53, 000.00 ) ;
```

1 record created.

## Example 4.2.93

```
insert into CLASS
values
( 024, 5, 503, 097.14, 085.72, 000.00 ) ;
```

1 record created.

## Example 4.2.94

```
insert into CLASS
values
( 025, 1, 101, 083.58, 089.16, 000.00 ) ;
```

1 record created.

## Example 4.2.95

And let's take a look at the data we inserted into the CLASS table.

```
select *
from CLASS ;
```

CLASS_STUDENT	CLASS_DEPT	CLASS_COURSE	CLASS_SEM_1	CLASS_SEM_2	CLASS_GRADE
1	3	302	89.49	51.91	.00
1	3	303	77.61	88.84	.00
2	1	103	54.38	84.77	.00
3	4	403	92.92	97.48	.00
4	2	204	71.17	70.55	.00
5	5	503	88.83	81.12	.00
6	3	301	66.26	94.60	.00
6	4	402	100.00	100.00	.00
7	4	401	100.00	100.00	.00
7	4	402	100.00	100.00	.00
7	4	403	100.00	100.00	.00
7	5	503	100.00	100.00	.00
8	5	502	69.68	56.92	.00
9	2	204	55.53	89.81	.00
10	1	102	93.72	99.55	.00
11	4	401	81.99	76.29	.00
12	5	501	75.81	83.03	.00

# UNCLASSIFIED

13	5	502	67.36	80.15	.00
14	3	302	92.27	82.47	.00
15	2	202	89.75	95.74	.00
16	1	101	85.64	78.26	.00
16	1	101	94.59	91.52	.00
16	2	204	83.40	94.88	.00
16	3	302	82.14	87.11	.00
16	4	403	89.92	97.40	.00
16	5	501	76.86	95.72	.00
17	4	401	94.71	63.36	.00
18	5	503	92.69	71.69	.00
19	2	201	81.31	95.95	.00
20	2	204	88.28	79.01	.00
21	3	303	71.16	74.14	.00
22	1	102	58.97	86.58	.00
22	2	201	81.75	92.97	.00
22	5	503	74.49	98.30	.00
23	4	402	96.33	81.53	.00
24	5	503	97.14	85.72	.00
25	1	101	83.58	89.16	.00

37 records selected.

## Example 4.2.96

Now we'll fill the SALARY table with data.

```
insert into SALARY
values
(1, 1, 20000.00, 29999.00, 0.010);
```

1 record created.

## Example 4.2.97

```
insert into SALARY
values
(2, 2, 30000.00, 34999.00, 0.075);
```

1 record created.

## Example 4.2.98

```
insert into SALARY
values
(3, 3, 35000.00, 39999.00, 0.050);
```

1 record created.

## Example 4.2.99



UNCLASSIFIED

```
insert into SALARY
values
(4, 4, 40000.00, 44999.00, 0.035);
```

1 record created.

**Example 4.2.100**

```
insert into SALARY
values
(5, 5, 45000.00, 49999.00, 0.025);
```

1 record created.

**Example 4.2.101**

```
insert into SALARY
values
(6, 10, 50000.00, 51999.00, 0.020);
```

1 record created.

**Example 4.2.102**

```
insert into SALARY
values
(11, 15, 52000.00, 53999.00, 0.020);
```

1 record created.

**Example 4.2.103**

```
insert into SALARY
values
(16, 20, 54000.00, 55999.00, 0.020);
```

1 record created.

**Example 4.2.104**

```
insert into SALARY
values
(21, 99, 56000.00, 60000.00, 0.020);
```

1 record created.

**Example 4.2.105**

And let's take a look at the information in the SALARY table.

```
select *
```

UNCLASSIFIED

from SALARY ;

SAL_YEAR	SAL_END	SAL_MIN	SAL_MAX	SAL_RAISE
1	1	20000.00	29999.00	.010
2	2	30000.00	34999.00	.075
3	3	35000.00	39999.00	.050
4	4	40000.00	44999.00	.035
5	5	45000.00	49999.00	.025
6	10	50000.00	51999.00	.020
11	15	52000.00	53999.00	.020
16	20	54000.00	55999.00	.020
21	99	56000.00	60000.00	.020

9 records selected.

We'll leave the GRADE table empty for now.

### 4.3 More Basic SELECTs

Before learning more query commands lets try a couple more selections of data.

#### Example 4.3.1

List all students, their first names, last names, room number and year.

```
select ST_FIRST, ST_NAME, ST_ROOM, ST_YEAR
from STUDENT ;
```

ST_FIRST	ST_NAME	ST_R	ST_YEAR
William	Horrigan	A101	4
Gregory	McGinn	A102	3
Molly	Lewis	A103	2
Dennis	Waxler	A104	2
Howard	McNamara	A201	1
Fay	Hess	A202	3
Jennifer	Guiffre	A203	1
Carl	Hagan	A204	4
Rose	Bearman	A301	1
Paul	Thompson	A302	3
Nellie	Bennett	A303	3
John	Schmidt	A304	2
Susan	Gevarter	B101	4
Donald	Sherman	B102	3
Milton	Gorham	B103	2
Alvin	Williams	B104	1
Dorothy	Woodliff	B201	4
Ann	Ratliff	B202	1
Kim	Phung	B203	2
Eric	McMurray	B204	1
Peggy	O'Leary	C101	4
Charlotte	Martin	C102	2
Hilda	O'Day	C103	1
Edward	Martin	C104	3
Chelsea	Chateauneuf	C105	3

25 records selected.

#### Example 4.3.2

Now give me a list of all the professors, their last names, salaries and number of years at the university.

```
select PROF_NAME, PROF_SALARY, PROF_YEARS
from PROFESSOR ;
```

PROF_NAME	PROF_SALARY	PROF_YEARS
Dysart	35000.00	3

## UNCLASSIFIED

Hall	45000.00	7
Steinbacner	30000.00	1
Bailey	50000.00	15
Clements	40000.00	4

### 4.4 DISTINCT

#### Example 4.4.1

I want a list of all states from which this year's students come from, listing only the home state without any identifying student information.

```
select ST_STATE
from STUDENT ;
```

ST  
VA  
MD  
PA  
NC  
VA  
DC  
MD  
PA  
VA  
NC  
PA  
SC  
NY  
VA  
WV  
DC  
MD  
NY  
SC  
VA  
PA  
DC  
NC  
MD  
VA

25 records selected.

We have a list containing several duplicate states. We want just a list of the home states with each state listed only once. To produce such a list we would use the distinct clause. This would list each distinctive state only once. If more than one column is selected when the distinct option is set then each record listed will not duplicate any other record listed. The format of the distinct clause is:

```
select distinct COLUMN_1, COLUMN_2, ...
```

UNCLASSIFIED

from TABLE ;

**Example 4.4.2**

List the states from which our students come, list each state only once.

```
select distinct ST_STATE
from STUDENT ;
```

ST  
DC  
MD  
NC  
NY  
PA  
SC  
VA  
WV

8 records selected.

**Example 4.4.3**

For each state represented by our student body do we have students attending their first, second, third or fourth year. I don't need to know the number of students in each category, only which categories exist in our student body.

```
select distinct ST_STATE, ST_YEAR
from STUDENT ;
```

ST	ST_YEAR
DC	1
DC	2
DC	3
MD	1
MD	3
MD	4
NC	1
NC	2
NC	3
NY	1
NY	4
PA	2
PA	3
PA	4
SC	2
VA	1
VA	3
VA	4

WV 2

19 records selected.

## 4.5 WHERE

You now know how to select all records or all distinctive records from a table. But frequently you will want to select only certain records based on specific criteria of one or more columns. The format of the where clause is:

```
select COLUMNS
from TABLES
where where_clause_comparison ;
```

## 4.6 WHERE With Comparison Operators = <> < <= > >=

The comparison operators available are:

equal to	=
not equal to	<>
less than	<
less than or equal to	<=
greater than	>
greater than or equal to	>=

Remember I'm using the ORACLE DBMS for these examples, the symbols used by your DBMS may vary. The format of the where clause with a comparison operator is:

```
select COLUMNS
from TABLES
where COLUMN comparison operator /COLUMN, constant/ ;
```

### Example 4.6.1

Let's select all students from Virginia.

```
select *
from STUDENT
where ST_STATE = 'VA' ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R ST	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101 VA	3	4
5	McNamara	Howard	A201 VA	5	1
9	Bearman	Rose	A301 VA	2	1
14	Sherman	Donald	B102 VA	3	3
20	McMurray	Eric	B204 VA	2	1
25	Chateauneuf	Chelsea	C105 VA	1	3

UNCLASSIFIED

6 records selected.

**Example 4.6.2**

Remember when comparisons are to character strings enclose the strings in quotes and pad with spaces. Select the student record for McGinn.

```
select *
from STUDENT
where ST_NAME = 'McGinn      ' ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R ST	ST_MAJOR	ST_YEAR
2	McGinn	Gregory	A102 MD	1	3

**Example 4.6.3**

List all professors who are teaching for the first year at our school.

```
select *
from PROFESSOR
where PROF_YEARS = 1 ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
3	Steinbacner	Moris	2	1	30000.00

**Example 4.6.4**

List all professors who are not teaching at the school for the first year.

```
select *
from PROFESSOR
where PROF_YEARS <> 1 ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

**Example 4.6.5**

Or we could list all professors who have taught at the school for more than one year. The same criteria stated differently.

```
select *
from PROFESSOR
where PROF_YEARS > 1 ;
```

# UNCLASSIFIED

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

## Example 4.6.6

List all professors who have taught at the school for at least four years.

```
select *
from PROFESSOR
where PROF_YEARS >= 4 ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
2	Hall	Elizabeth	4	7	45000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

## Example 4.6.7

List all professors teaching for under four years.

```
select *
from PROFESSOR
where PROF_YEARS < 4 ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
3	Steinbacner	Moris	2	1	30000.00

## Example 4.6.8

Or we could phrase it as, list all professors who have been with the school no more than three years.

```
select *
from PROFESSOR
where PROF_YEARS <= 3 ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
3	Steinbacner	Moris	2	1	30000.00

## Example 4.6.9

We can compare one column to another, for example, list all classes where the students grades for the second semester were lower than their grades for the first semester. This compares the specified columns from the same record with each other. It will not compare columns from different records.



# UNCLASSIFIED

```
select *
from CLASS
where CLASS_SEM_2 < CLASS_SEM_1 ;
```

CLASS_STUDENT	CLASS_DEPT	CLASS_COURSE	CLASS_SEM_1	CLASS_SEM_2	CLASS_GRADE
1	3	302	89.49	51.91	.00
4	2	204	71.17	70.55	.00
5	5	503	88.83	81.12	.00
8	5	502	69.68	56.92	.00
11	4	401	81.99	76.29	.00
14	3	302	92.27	82.47	.00
16	1	101	85.64	78.26	.00
16	1	101	94.59	91.52	.00
17	4	401	94.71	63.36	.00
18	5	503	92.69	71.69	.00
20	2	204	88.28	79.01	.00
23	4	402	96.33	81.53	.00
24	5	503	97.14	85.72	.00

13 records selected.

## 4.7 WHERE With AND & OR

You can create rather complex selection criteria by adding the use of ANDs and ORs and selecting the precedence of the operators with parenthesis.

### Example 4.7.1

Select all students from Virginia here for the first year.

```
select *
from STUDENT
where ST_STATE = 'VA' and ST_YEAR = 1 ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R ST	ST_MAJOR	ST_YEAR
5	McNamara	Howard	A201 VA	5	1
9	Bearman	Rose	A301 VA	2	1
20	McMurray	Eric	B204 VA	2	1

### Example 4.7.2

List all students from North or South Carolina.

```
select *
from STUDENT
where ST_STATE = 'NC' or ST_STATE = 'SC' ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R ST	ST_MAJOR	ST_YEAR
4	Waxler	Dennis	A104 NC	2	2

# UNCLASSIFIED

10	Thompson Paul	A302 NC	1	3
12	Schmidt John	A304 SC	5	2
19	Phung Kim	B203 SC	2	2
23	O'Day Hilda	C103 NC	4	1

Note how we have to list the column ST\_STATE twice and cannot simply request ST\_STATE = 'NC' or 'SC'. ANDs and ORs must link complete comparisons not just the comparison values.

## Example 4.7.3

List all students from North or South Carolina and in their second year.

```
select *
from STUDENT
where ( ST_STATE = 'NC' or ST_STATE = 'SC' )
and ST_YEAR = 2 ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R ST	ST_MAJOR	ST_YEAR
4	Waxler	Dennis	A104 NC	2	2
12	Schmidt	John	A304 SC	5	2
19	Phung	Kim	B203 SC	2	2

## Example 4.7.4

List all professors who have taught for four years or less and earn a salary of more than \$33,000.00.

```
select *
from PROFESSOR
where PROF_YEARS <= 4
and PROF_SALARY > 33000.00 ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
5	Clements	Carol	1	4	40000.00

We can generate quite complicated queries by combining multiple ANDs and ORs.

## Example 4.7.5

List all students from Virginia in their first year, all students from North or South Carolina in their second year, all students from Maryland in their third year, all students in their fourth year and all students from the District of Columbia.

```
select *
from STUDENT
where ( ST_STATE = 'VA' and ST_YEAR = 1 )
or ( ( ST_STATE = 'NC' or ST_STATE = 'SC' )
and ST_YEAR = 2 )
or ( ST_STATE = 'MD' and ST_YEAR = 3 )
```

# UNCLASSIFIED

```
or ST_YEAR = 4
or ST_STATE = 'DC' ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	4
2	McGinn	Gregory	A102	MD	1	3
4	Waxler	Dennis	A104	NC	2	2
5	McNamara	Howard	A201	VA	5	1
6	Hess	Fay	A202	DC	3	3
8	Hagan	Carl	A204	PA	5	4
9	Bearman	Rose	A301	VA	2	1
12	Schmidt	John	A304	SC	5	2
13	Gevarter	Susan	B101	NY	5	4
16	Williams	Alvin	B104	DC	1	1
17	Woodliff	Dorothy	B201	MD	4	4
19	Phung	Kim	B203	SC	2	2
20	McMurray	Eric	B204	VA	2	1
21	O'Leary	Peggy	C101	PA	3	4
22	Martin	Charoltte	C102	DC	1	2
24	Martin	Edward	C104	MD	5	3

16 records selected.

## 4.8 WHERE With BETWEEN Operator

If you want to select information based on a range of values you could do it with a complex query.

### Example 4.8.1

Let's list all professors who's salaries fall into a range of from \$35,000 to \$45,000.

```
select *
from PROFESSOR
where PROF_SALARY >= 35000.00
and PROF_SALARY <= 45000.00 ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
5	Clements	Carol	1	4	40000.00

The between comparison operator allows you to select values in a range in a less awkward fashion. The format for the between operator is:

```
select COLUMNS
from TABLES
where COLUMN between LO_LIMIT and HI_LIMIT ;
```

UNCLASSIFIED

**Example 4.8.2**

List all professors who's salaries fall into a range of from \$35,000 to \$45,000, using the between operator.

```
select *  
  from Professor  
   where PROF_SALARY  
         between 35000.00 and 45000.00 ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
5	Clements	Carol	1	4	40000.00

#### 4.9 WHERE With The IN Operator

There are times when we may want to select column information from a list of possible constants. We could do this using ORs.

##### Example 4.9.1

For example select all students from Virginia, Maryland and the District of Columbia.

```
select *
  from STUDENT
    where ST_STATE = 'VA'
       or ST_STATE = 'MD'
       or ST_STATE = 'DC' ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	4
2	McGinn	Gregory	A102	MD	1	3
5	McNamara	Howard	A201	VA	5	1
6	Hess	Fay	A202	DC	3	3
7	Guiffre	Jennifer	A203	MD	4	1
9	Bearman	Rose	A301	VA	2	1
14	Sherman	Donald	B102	VA	3	3
16	Williams	Alvin	B104	DC	1	1
17	Woodliff	Dorothy	B201	MD	4	4
20	McMurray	Eric	B204	VA	2	1
22	Martin	Charoltte	C102	DC	1	2
24	Martin	Edward	C104	MD	5	3
25	Chateauneuf	Chelsea	C105	VA	1	3

13 records selected.

Or we could simplify this query by using the in operator, which allows us to select a column if its contents are equal to one item in a group. The format for the in operator is:

```
select COLUMNS
  from TABLES
    where COLUMN in ( OPTION_1, OPTION_2, ... ) ;
```

##### Example 4.9.2

So the above example could be shortened to:

```
select *
  from STUDENT
    where ST_STATE in ( 'VA', 'MD', 'DC' ) ;
```

ST ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	4

## UNCLASSIFIED

2	McGinn	Gregory	A102	MD	1	3
5	McNamara	Howard	A201	VA	5	1
6	Hess	Fay	A202	DC	3	3
7	Guiffre	Jennifer	A203	MD	4	1
9	Bearman	Rose	A301	VA	2	1
14	Sherman	Donald	B102	VA	3	3
16	Williams	Alvin	B104	DC	1	1
17	Woodliff	Dorothy	B201	MD	4	4
20	McMurray	Eric	B204	VA	2	1
22	Martin	Charlotte	C102	DC	1	2
24	Martin	Edward	C104	MD	5	3
25	Chateauneuf	Chelsea	C105	VA	1	3

13 records selected.

### 4.10 Wild Characters

Wild characters are special characters used for the purpose of comparison with character strings. A percent symbol % matches any sequence of zero or more characters. An underscore \_ matches any one character. For example A% would match any character string, regardless of its length, if it began with the character A. A\_CDE would match an character string, five characters long, where the first character was A and the third, fourth and fifth were CDE and the second was any character.

### 4.11 WHERE With LIKE Operator

When comparing column values to constants with comparison operators or the in operator the constant must match the data in the column exactly. But there may be times when you only want to match parts of column data using the wild characters described above. The like operator is used when you wish to use pattern matching. The format of the like clause is:

```
select COLUMNS
from TABLES
where COLUMN like pattern_matching_string ;
```

#### Example 4.11.1

To search for all names beginning with S you would use the pattern matching string 'S\*'.

```
select ST_NAME
from STUDENT
where ST_NAME like 'S%' ;
```

```
ST_NAME
Schmidt
Sherman
```

UNCLASSIFIED

**Example 4.11.2**

To search for all students in dorm building A you would use the pattern matching string 'A\*'.

```
select ST_NAME, ST_ROOM
from STUDENT
where ST_ROOM like 'A*' ;
```

ST_NAME	ST_R
Horrigan	A101
McGinn	A102
Lewis	A103
Waxler	A104
McNamara	A201
Hess	A202
Guiffre	A203
Hagan	A204
Bearman	A301
Thompson	A302
Bennett	A303
Schmidt	A304

12 records selected.

**Example 4.11.3**

Or the pattern matching string 'A\_\_'.

```
select ST_NAME, ST_ROOM
from STUDENT
where ST_ROOM like 'A__' ;
```

ST_NAME	ST_R
Horrigan	A101
McGinn	A102
Lewis	A103
Waxler	A104
McNamara	A201
Hess	A202
Guiffre	A203
Hagan	A204
Bearman	A301
Thompson	A302
Bennett	A303
Schmidt	A304

12 records selected.

**Example 4.11.4**

## UNCLASSIFIED

To search for all students in room 101 of any dorm building you could use the pattern matching string '\_101'.

```
select ST_NAME, ST_ROOM
from STUDENT
where ST_ROOM like '_101' ;
```

ST_NAME	ST_R
Horrigan	A101
Gevarter	B101
O'Leary	C101

### 4.12 WHERE With The NOT Operator

There may be times you wish to select all records except those matching certain criteria. The not operator would be used here. It can be used with the comparison operators ( =, <>, <, <=, >, >= ), with the between operator, the in operator and the like operator. When using the comparison operators the equation must be surrounded by parenthesis. The format of the not operator is the same as the other operators but with the word not in front of it.

```
select COLUMNS
from TABLES
where not where_clause_comparison ;
```

The format of the where clause with comparison operators is :

```
select COLUMNS
from TABLES
where not ( COLUMN comparison operator /COLUMN, constant/ ) ;
```

The format of the where clause with the between operator is:

```
select COLUMNS
from TABLES
where COLUMN not between LO_LIMIT and HI_LIMIT ;
```

The format of the where clause with the in operator is:

```
select COLUMNS
from TABLES
where COLUMN not in ( OPTION_1, OPTION_2, ... ) ;
```

The format of the where clause with the like operator is:

```
select COLUMNS
from TABLES
where COLUMN not like pattern_matching_string ;
```

Here are several examples of the not operator.



# UNCLASSIFIED

## Example 4.12.1

Select all students who are not from Virginia.

```
select *
from STUDENT
where not (ST_STATE = 'VA' ) ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
2	McGinn	Gregory	A102	MD	1	3
3	Lewis	Molly	A103	PA	4	2
4	Waxler	Dennis	A104	NC	2	2
6	Hess	Fay	A202	DC	3	3
7	Guiffre	Jennifer	A203	MD	4	1
8	Hagan	Carl	A204	PA	5	4
10	Thompson	Paul	A302	NC	1	3
11	Bennett	Nellie	A303	PA	4	3
12	Schmidt	John	A304	SC	5	2
13	Gevarter	Susan	B101	NY	5	4
15	Gorham	Milton	B103	WV	2	2
16	Williams	Alvin	B104	DC	1	1
17	Woodliff	Dorothy	B201	MD	4	4
18	Ratliff	Ann	B202	NY	5	1
19	Phung	Kim	B203	SC	2	2
21	O'Leary	Peggy	C101	PA	3	4
22	Martin	Charoltte	C102	DC	1	2
23	O'Day	Hilda	C103	NC	4	1
24	Martin	Edward	C104	MD	5	3

19 records selected.

## Example 4.12.2

List all professors except those who have taught for four years or less and earn a salary of more than \$33,000.00.

```
select *
from PROFESSOR
where not ( PROF_YEARS <= 4
and PROF_SALARY > 33000.00 ) ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00

## Example 4.12.3

List all professors who's salaries fall outside a range of from \$35,000 to \$45,000.

# UNCLASSIFIED

```
select *
  from Professor
   where PROF_SALARY
        not between 35000.00 and 45000.00 ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00

## Example 4.12.4

Select all students from anywhere except Virginia, Maryland and the District of Columbia.

```
select *
  from STUDENT
   where ST_STATE not in ( 'VA', 'MD', 'DC' ) ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
3	Lewis	Molly	A103	PA	4	2
4	Waxler	Dennis	A104	NC	2	2
8	Hagan	Carl	A204	PA	5	4
10	Thompson	Paul	A302	NC	1	3
11	Bennett	Nellie	A303	PA	4	3
12	Schmidt	John	A304	SC	5	2
13	Gevarter	Susan	B101	NY	5	4
15	Gorham	Milton	B103	WV	2	2
18	Ratliff	Ann	B202	NY	5	1
19	Phung	Kim	B203	SC	2	2
21	O'Leary	Peggy	C101	PA	3	4
23	O'Day	Hilda	C103	NC	4	1

12 records selected.

## Example 4.12.5

Search for all students in all dorm buildings except building A.

```
select ST_NAME, ST_ROOM
  from STUDENT
   where ST_ROOM not like 'A%' ;
```

ST_NAME	ST_ROOM
Gevarter	B101
Sherman	B102
Gorham	B103
Williams	B104
Woodliff	B201
Ratliff	B202
Phung	B203

## UNCLASSIFIED

McMurray	B204
O'Leary	C101
Martin	C102
O'Day	C103
Martin	C104
Chateauneuf	C105

13 records selected.

### 4.13 The Arithmetic Expressions + - \* /

You may wish to perform arithmetic calculations on the data in numeric columns for display purposes or for the purpose of comparison. You can use an arithmetic expression by connecting numeric columns and or numeric constants with the arithmetic operators:

- + add
- subtract
- \* multiply
- / divide

You may use parenthesis to establish precedence of operations within an expression. Arithmetic expressions may be used wherever a column name is allowed. An arithmetic operation may be performed between one or more numeric fields and/or using one or more numeric constants.

An arithmetic expression may be used in place of a column name in the list of columns to select. Why would you want to do this? Imagine you'd like to see what salaries would be if everyone got an 10% raise, but you don't really want to change the data. So you would select salary \* 1.10. The format of an arithmetic expression as an item in a select list is:

```
select column_or_constant arithmetic_operator column_or_operator ...
```

#### Example 4.13.1

List the professors and their salaries if they were to receive a 10% raise.

```
select PROF_NAME, PROF_SALARY * 1.10
from PROFESSOR ;
```

PROF_NAME	PROF_SALARY*1.10
Dysart	38500.00
Hall	49500.00
Steinbacner	33000.00
Bailey	55000.00
Clements	44000.00

#### Example 4.13.2

List the average of the two semester grades for the classes without using the CLASS\_GRADE field, use

# UNCLASSIFIED

only the semester grades.

```
select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
from CLASS ;
```

CLASS_STUDENT	(CLASS_SEM_1+CLASS_SEM_2)/2
1	70.70
1	83.23
2	69.58
3	95.20
4	70.86
5	84.98
6	80.43
6	100.00
7	100.00
7	100.00
7	100.00
7	100.00
8	63.30
9	72.67
10	96.64
11	79.14
12	79.42
13	73.76
14	87.37
15	92.75
16	81.95
16	93.06
16	89.14
16	84.63
16	93.66
16	86.29
17	79.04
18	82.19
19	88.63
20	83.65
21	72.65
22	72.78
22	87.36
22	86.40
23	88.93
24	91.43
25	86.37

37 records selected.

An arithmetic expression may be used in place of a column name as selection criteria in a where clause. For example in a table which included current salary and previous years salary you might want to select anyone who's salary is greater than 10% more than last years salary. The format of an arithmetic expression in a where clause is:

## UNCLASSIFIED

where column\_or\_constant arithmetic\_operator column\_or\_operator ...

### Example 4.13.3

List the professors who have a salary which is greater than \$10,000 for each year of their employment.

```
select PROF_NAME, PROF_SALARY
from PROFESSOR
where PROF_SALARY > PROF_YEARS * 10000.00 ;
```

PROF_NAME	PROF_SALARY
Dysart	35000.00
Steinbacner	30000.00

### Example 4.13.4

Now if we were to give our professors a 10% raise which ones would be making less than \$10000.00 for each year of employment.

```
select PROF_NAME, PROF_SALARY * 1.10
from PROFESSOR
where PROF_SALARY * 1.10 < PROF_YEARS * 10000.00 ;
```

PROF_NAME	PROF_SALARY*1.10
Hall	49500.00
Bailey	55000.00

## 4.14 The Aggregate Functions COUNT, MIN, MAX, SUM, AVG

The aggregate functions are applied to all selected records in a table and provide information using data from each record. These functions return summary information about groups of records in the table. For example we can list a count of the student body, the minimum or maximum salary paid to a professor, the totally expenditure in salaries, the average class grades for a student. These functions may be applied to column names in the list of columns to select. The format of the aggregate functions is:

```
aggregate_function ( column_name )
```

The aggregate functions available are:

- COUNT - the number of values in the column chosen
- MIN - the minimum value in the column chosen
- MAX - the maximum value in the column chosen
- SUM - the total of the values in the column chosen
- AVG - the average of the values in the column chosen

Only one row will be listed as output and any column selected to be listed must apply to all records to be selected as part of the group selected. For example if you were to list a count of the student body

## UNCLASSIFIED

it would be inappropriate to request that student name be listed. However if you were to list minimum salary for professors you could request the name also. But what would happen if more than one professor earned the same salary which was the lowest salary? Your data would be erroneous since only one name would be listed. So please, to avoid confusion, list only data which applies to all records that might be used to create the requested information.

When calculating an aggregate function first all records are selected based on the criteria in the where clause. Then the function is applied to the aggregate fields and one row of information is displayed. Aggregate functions may be used only in a select clause (or a having clause which we will cover later on) and may never be used as selection criteria in a where clause. Remember that these functions will display one and only one row of information which is an aggregate of all the records selected based on criteria in the where clause.

### Example 4.14.1

List the count of the student body, which would simple be a count of the records in the STUDENT table, since each student gets one and only one record.

```
select count (*)  
  from STUDENT ;
```

```
COUNT(*)  
      25
```

Note the heading displayed above. The aggregate function as it is in the select clause will be used as the heading for the column of displayed information.

### Example 4.14.2

Now count the number of students in dorm building A.

```
select count (*)  
  from STUDENT  
   where ST_ROOM like 'A%';
```

```
COUNT(*)  
      12
```

### Example 4.14.3

Count the number of students from the Washington DC area, include Virginia and Maryland.

```
select count (*)  
  from STUDENT  
   where ST_STATE in ( 'DC', 'VA', 'MD' );
```

```
COUNT(*)  
      13
```

**Example 4.14.4**

Now count the number of students from Virginia and list the state being counted. I'm assuming it's ok to list the state here since all states being selected contain the same information in the column being listed.

```
select ST_STATE, count (*)
  from STUDENT
 where ST_STATE = 'VA' ;
```

ERROR at line 1: ORA-0937: not a single group set function

Oops, I guess it's not ok. The DBMS is not going to allow us to list a column which could result in erroneous results. The DBMS is disallowing this query since if we were counting all records, the values for state would not be the same in all records. It is not taking in consideration the fact that we have narrowed the selection to only states of VA. Some DBMSs may permit such a query.

**Example 4.14.5**

Let's try the query without the offending column selection.

```
select count (*)
  from STUDENT
 where ST_STATE = 'VA' ;
```

```
COUNT(*)
      6
```

**Example 4.14.6**

The CLASS table lists each class taken by each student. I want a count of the unique classes being taken by all the students. I can use the distinct function in conjunction with an aggregate function to obtain the answer.

```
select count ( distinct CLASS_COURSE )
  from CLASS ;
```

```
COUNT(DISTINCTCLASS_COURSE)
                        15
```

**Example 4.14.7**

Now let's list the minimum and maximum salary paid to the professors.

```
select min (PROF_SALARY), max (PROF_SALARY)
  from PROFESSOR ;
```

UNCLASSIFIED

MIN(PROF_SALARY)	MAX(PROF_SALARY)
30000.00	50000.00

**Example 4.14.8**

Add all professor's salaries together to list the total salary expenditure.

```
select sum (PROF_SALARY)
  from PROFESSOR ;
```

SUM(PROF_SALARY)
200000.00

**Example 4.14.9**

Now list the average first and second semester grades for all classes taken by student Alvin Williams who is student number 016.

```
select avg (CLASS_SEM_1), avg (CLASS_SEM_2)
  from CLASS
  where CLASS_STUDENT = 016 ;
```

AVG(CLASS_SEM_1)	AVG(CLASS_SEM_2)
85.43	90.82

**Example 4.14.10**

List the number of professors at our school, our total salary expenditure, the average salary per professor and the minimum and maximum salaries paid.

```
select count (*), sum (PROF_SALARY), avg (PROF_SALARY),
  min (PROF_SALARY), max (PROF_SALARY)
  from PROFESSOR ;
```

COUNT(*)	SUM(PROF_SALARY)	AVG(PROF_SALARY)	MIN(PROF_SALARY)	MAX(PROF_SALARY)
5	200000.00	40000.00	30000.00	50000.00



# UNCLASSIFIED

## 4.15 ORDER BY

So far we've just asked to see data and not paid any attention to the order in which the output was displayed. You will usually want to order your lists in some way. You can do this using the order by clause. The format of an order by clause is:

```
order by COLUMN sequence, COLUMN sequence, ...
```

Multiple columns may be listed, each subsequent column will be sorted as a subset of the previous column. Each column may specify if the sort sequence is to be ascending, ASC, or descending, DESC. Ascending, from smallest such as A or 1 to the largest, Z or 9, is the default.

### Example 4.15.1

List all students in order of last name.

```
select *
  from STUDENT
  order by ST_NAME ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R ST	ST_MAJOR	ST_YEAR
9	Bearman	Rose	A301 VA	2	1
11	Bennett	Nellie	A303 PA	4	3
25	Chateauneuf	Chelsea	C105 VA	1	3
13	Gevarter	Susan	B101 NY	5	4
15	Gorham	Milton	B103 WV	2	2
7	Guiffre	Jennifer	A203 MD	4	1
8	Hagan	Carl	A204 PA	5	4
6	Hess	Fay	A202 DC	3	3
1	Horrigan	William	A101 VA	3	4
3	Lewis	Molly	A103 PA	4	2
22	Martin	Charoltte	C102 DC	1	2
24	Martin	Edward	C104 MD	5	3
2	McGinn	Gregory	A102 MD	1	3
20	McMurray	Eric	B204 VA	2	1
5	McNamara	Howard	A201 VA	5	1
23	O'Day	Hilda	C103 NC	4	1
21	O'Leary	Peggy	C101 PA	3	4
19	Phung	Kim	B203 SC	2	2
18	Ratliff	Ann	B202 NY	5	1
12	Schmidt	John	A304 SC	5	2
14	Sherman	Donald	B102 VA	3	3
10	Thompson	Paul	A302 NC	1	3
4	Waxler	Dennis	A104 NC	2	2
16	Williams	Alvin	B104 DC	1	1
17	Woodliff	Dorothy	B201 MD	4	4

25 records selected.

# UNCLASSIFIED

## Example 4.15.2

List all professors and their salaries with the largest salary first.

```
select PROF_NAME, PROF_SALARY
from PROFESSOR
order by PROF_SALARY desc ;
```

PROF_NAME	PROF_SALARY
Bailey	50000.00
Hall	45000.00
Clements	40000.00
Dysart	35000.00
Steinbacner	30000.00

## Example 4.15.3

List all students by the number of years they have studied and the major they are studying. List the students with the most number of years first.

```
select ST_NAME, ST_YEAR, ST_MAJOR
from STUDENT
order by ST_YEAR desc, ST_MAJOR ;
```

ST_NAME	ST_YEAR	ST_MAJOR
Horrigan	4	3
O'Leary	4	3
Woodliff	4	4
Hagan	4	5
Gevarter	4	5
McGinn	3	1
Chateauneuf	3	1
Thompson	3	1
Hess	3	3
Sherman	3	3
Bennett	3	4
Martin	3	5
Martin	2	1
Waxler	2	2
Gorham	2	2
Phung	2	2
Lewis	2	4
Schmidt	2	5
Williams	1	1
Bearman	1	2
McMurray	1	2
Guiffre	1	4
O'Day	1	4
McNamara	1	5
Ratliff	1	5

# UNCLASSIFIED

25 records selected.

## Example 4.15.4

When sorting in ascending order I have omitted the ASC specification since it is the default. To include it in the above example we'd use:

```
select ST_NAME, ST_YEAR, ST_MAJOR
from STUDENT
order by ST_YEAR desc, ST_MAJOR asc ;
```

ST_NAME	ST_YEAR	ST_MAJOR
Horrigan	4	3
O'Leary	4	3
Woodliff	4	4
Hagan	4	5
Gevarter	4	5
McGinn	3	1
Chateauneuf	3	1
Thompson	3	1
Hess	3	3
Sherman	3	3
Bennett	3	4
Martin	3	5
Martin	2	1
Waxler	2	2
Gorham	2	2
Phung	2	2
Lewis	2	4
Schmidt	2	5
Williams	1	1
Bearman	1	2
McMurray	1	2
Guiffre	1	4
O'Day	1	4
McNamara	1	5
Ratliff	1	5

25 records selected.

## Example 4.15.5

For each class list the department, course, average for both semesters, and the student's id. List only those class where the average of the grade for the first semester and second semester is at least 90% and the grade of the second semester is at least 5% better than the grade for the first semester or where the student got a grade of 100% in either the first or second semester. Sort the list in the order of the highest second semester grade, highest first semester grade the department and course.

```
select CLASS_DEPT, CLASS_COURSE, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2,
```

## UNCLASSIFIED

```
CLASS_STUDENT
from CLASS
where ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 >= 90.00
and CLASS_SEM_2 - CLASS_SEM_1 >= 5.00 )
or CLASS_SEM_1 = 100.00
or CLASS_SEM_2 = 100.00
order by CLASS_SEM_2 desc, CLASS_SEM_1 desc, CLASS_DEPT asc,
CLASS_COURSE asc ;
```

CLASS_DEPT	CLASS_COURSE	(CLASS_SEM_1+CLASS_SEM_2)/2	CLASS_STUDENT
4	401	100.00	7
4	402	100.00	6
4	402	100.00	7
4	403	100.00	7
5	503	100.00	7
1	102	96.64	10
4	403	93.66	16
2	202	92.75	15

8 records selected.

Let me point out here that I have introduced the clauses, select, from, where, and order by in the order in which they must appear if they are to be in a query. An order by clause may never be followed by a from clause etc.

### 4.16 GROUP BY

In one of the previous examples we wanted to know the average grades for all classes taken by one student. To get a list of the average grades for all students we would have to write a separate query for each student. How cumbersome. But there is a better way. We could use the group by clause which is used in conjunction with an aggregate function to perform a computation on common groups of records. In the past we used aggregate functions which treated all selected records as one group. By using the group by you may split your selected records into several groups and perform aggregate functions on each of those groups. The selected records are sorted and group breaks are made for each column in the group by clause. For each record listed the columns in the group by clause will be unique and the aggregate totals will be listed. The group by is always used in conjunction with an aggregate function. It has no meaning in a query without an aggregate function. The format of the group by clause is:

```
group by COLUMN, COLUMN, ...
```

#### Example 4.16.1

For example to list the average first and second semester grades for all classes taken by all students we would use the following query.

```
select CLASS_STUDENT, avg (CLASS_SEM_1), avg (CLASS_SEM_2)
```

# UNCLASSIFIED

```
from CLASS
group by CLASS_STUDENT ;
```

CLASS_STUDENT	AVG(CLASS_SEM_1)	AVG(CLASS_SEM_2)
1	83.55	70.38
2	54.38	84.77
3	92.92	97.48
4	71.17	70.55
5	88.83	81.12
6	83.13	97.30
7	100.00	100.00
8	69.68	56.92
9	55.53	89.81
10	93.72	99.55
11	81.99	76.29
12	75.81	83.03
13	67.36	80.15
14	92.27	82.47
15	89.75	95.74
16	85.43	90.82
17	94.71	63.36
18	92.69	71.69
19	81.31	95.95
20	88.28	79.01
21	71.16	74.14
22	71.74	92.62
23	96.33	81.53
24	97.14	85.72
25	83.58	89.16

25 records selected.

## Example 4.16.2

If we wanted the same listing but only for department 3 we would use the where clause in the following query.

```
select CLASS_STUDENT, avg (CLASS_SEM_1), avg (CLASS_SEM_2)
from CLASS
where CLASS_DEPT = 3
group by CLASS_STUDENT ;
```

CLASS_STUDENT	AVG(CLASS_SEM_1)	AVG(CLASS_SEM_2)
1	83.55	70.38
6	66.26	94.60
14	92.27	82.47
16	82.14	87.11
21	71.16	74.14

# UNCLASSIFIED

## Example 4.16.3

Or if we wanted the listing of the student's grades but also broken down by department we would the following query.

```
select CLASS_STUDENT, CLASS_DEPT, avg (CLASS_SEM_1), avg (CLASS_SEM_2)
from CLASS
group by CLASS_DEPT, CLASS_STUDENT ;
```

CLASS_STUDENT	CLASS_DEPT	AVG(CLASS_SEM_1)	AVG(CLASS_SEM_2)
2	1	54.38	84.77
10	1	93.72	99.55
16	1	90.12	84.89
22	1	58.97	86.58
25	1	83.58	89.16
4	2	71.17	70.55
9	2	55.53	89.81
15	2	89.75	95.74
16	2	83.40	94.88
19	2	81.31	95.95
20	2	88.28	79.01
22	2	81.75	92.97
1	3	83.55	70.38
6	3	66.26	94.60
14	3	92.27	82.47
16	3	82.14	87.11
21	3	71.16	74.14
3	4	92.92	97.48
6	4	100.00	100.00
7	4	100.00	100.00
11	4	81.99	76.29
16	4	89.92	97.40
17	4	94.71	63.36
23	4	96.33	81.53
5	5	88.83	81.12
7	5	100.00	100.00
8	5	69.68	56.92
12	5	75.81	83.03
13	5	67.36	80.15
16	5	76.86	95.72
18	5	92.69	71.69
22	5	74.49	98.30
24	5	97.14	85.72

33 records selected.

## Example 4.16.4

List the number of students from each state, studying each major and in each year of study.

UNCLASSIFIED

```
select ST_STATE, ST_MAJOR, ST_YEAR, count(*)
from STUDENT
group by ST_STATE, ST_MAJOR, ST_YEAR ;
```

ST	ST_MAJOR	ST_YEAR	COUNT(*)
DC	1	1	1
DC	1	2	1
DC	3	3	1
MD	1	3	1
MD	4	1	1
MD	4	4	1
MD	5	3	1
NC	1	3	1
NC	2	2	1
NC	4	1	1
NY	5	1	1
NY	5	4	1
PA	3	4	1
PA	4	2	1
PA	4	3	1
PA	5	4	1
SC	2	2	1
SC	5	2	1
VA	1	3	1
VA	2	1	2
VA	3	3	1
VA	3	4	1
VA	5	1	1
WV	2	2	1

24 records selected.

#### 4.17 Nested Queries

##### Example 4.17.1

If we wanted to find out which professor made the highest salary, without having to look at the salaries for each one, we'd have to enter a query to select the maximum salary from the professor table, such as:

```
select max (PROF_SALARY)
from PROFESSOR ;
```

```
MAX(PROF_SALARY)
50000.00
```

##### Example 4.17.2

## UNCLASSIFIED

Then we'd have to enter a second query to list the professors who earn \$50,000, which is what we discovered to be the maximum salary in the last query.

```
select PROF_FIRST, PROF_NAME, PROF_SALARY
from PROFESSOR
where PROF_SALARY = 50000.00 ;
```

PROF_FIRST	PROF_NAME	PROF_SALARY
Bruce	Bailey	50000.00

We would have to get the desired information with two queries since we cannot use aggregate functions in the where clause. However these two queries can be nested together to produce the same result.

Nested queries are used when you want to select records from a table using selection criteria contained within that same table. In the above example we needed to know the maximum salary before we could select all records with the maximum salary. When nesting queries you use the result of one query as the selection criteria for the next query. A nested query may be a part of the where clause of a query. A nested query is called a subquery of the query in which it is nested. The simplest form of a subquery returns only one value. For example the above subquery returned the maximum salary. The format of a simple one value subquery is:

```
select COLUMN
from TABLE
where COLUMN_CONDITIONS operator
( select COLUMN
from TABLE
where COLUMN_CONDITIONS operator ) ... ;
```

A subquery of this format must return only one record or value. If more than one record is selected in the subquery the DBMS will signal an error. Queries may be nested to any level with the results of the deepest one being the conditions for the next.

### Example 4.17.3

To do the above queries as one nested query we would list the professor's names who earn the maximum salary of all professors.

```
select PROF_FIRST, PROF_NAME, PROF_SALARY
from PROFESSOR
where PROF_SALARY =
( select max (PROF_SALARY)
from PROFESSOR ) ;
```

PROF_FIRST	PROF_NAME	PROF_SALARY
Bruce	Bailey	50000.00

Multiple subqueries may be linked together in the where clause by using ANDs and/or ORs.



**Example 4.17.4**

For example list professor id and salary for all professors who are not earning the minimum or the maximum salary.

```
select PROF_ID, PROF_SALARY
  from PROFESSOR
 where PROF_SALARY >
    ( select min ( PROF_SALARY )
      from PROFESSOR )
 and PROF_SALARY <
    ( select max ( PROF_SALARY )
      from PROFESSOR ) ;
```

PROF_ID	PROF_SALARY
1	35000.00
2	45000.00
5	40000.00

**Example 4.17.5**

A subquery may also have a subquery of it's own. For example list the id, salary and number of years of employment for the professors earning more than the minimum salary of professors who have served more than the average number of years.

```
select PROF_ID, PROF_SALARY, PROF_YEARS
  from PROFESSOR
 where PROF_SALARY >
    ( select min ( PROF_SALARY )
      from PROFESSOR
      where PROF_YEARS >
        ( select avg ( PROF_YEARS )
          from PROFESSOR ) ) ;
```

PROF_ID	PROF_SALARY	PROF_YEARS
4	50000.00	15

Subqueries may also be written to return a set of values instead of only one value. The where clause must specify how the values returned are to be treated. This is specified by using the keyword ANY or ALL between the comparison operator and the subquery in the where clause. When using the keyword any, if the comparison to any of the values selected in the subquery is true then the record is selected. When using the keyword all, the comparison to each of the values selected in the subquery must be true in order for the record to be selected. The format of a multi value subquery is:

```
select COLUMN
  from TABLE
 where COLUMN_CONDITIONS operator and/all
    ( select COLUMN
      from TABLE
      where COLUMN_CONDITIONS operator ) ;
```

# UNCLASSIFIED

## Example 4.17.6

For example select any student who is taking more than two classes. List the student's id and average grade.

```
select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
from CLASS
where CLASS_STUDENT = any
( select CLASS_STUDENT
  from CLASS
    group by CLASS_STUDENT
  having count (*) > 2 ) ;
```

CLASS_STUDENT	(CLASS_SEM_1+CLASS_SEM_2)/2
7	100.00
7	100.00
7	100.00
7	100.00
16	81.95
16	93.66
16	86.29
16	84.63
16	89.14
16	93.06
22	72.78
22	86.40
22	87.36

13 records selected.

## Example 4.17.7

Select all the students and their average grade for all classes where the student's grade is greater than or equal to all grades earned by all students in all classes. In other words select the highest grades earned.

```
select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
from CLASS
where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 >= all
( select ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
  from CLASS ) ;
```

CLASS_STUDENT	(CLASS_SEM_1+CLASS_SEM_2)/2
6	100.00
7	100.00
7	100.00
7	100.00
7	100.00
7	100.00

## UNCLASSIFIED

### Example 4.17.8

You must be careful when deciding if you should use the keyword any or all. For example if we were to select students equal to all the students taking more than two classes we would select no records.

```
select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
  from CLASS
  where CLASS_STUDENT = all
( select CLASS_STUDENT
  from CLASS
  group by CLASS_STUDENT
  having count (*) > 2 ) ;
```

no records selected

Why do we get no records? Because our column CLASS\_STUDENT will be compared to all of the CLASS\_STUDENT columns from the subquery and must be equal to all of them in order to be selected. If more than one student is taking more than two classes the column in the record which we are checking cannot be equal to all values for the column selected in the subquery. Therefore every record is rejected.

### Example 4.17.9

Likewise if we were to select the class grades where the average grade was greater than or equal to any other grade in the table we'd end up selecting all records.

```
select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
  from CLASS
  where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 >= any
( select ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
  from CLASS ) ;
```

CLASS_STUDENT	(CLASS_SEM_1+CLASS_SEM_2)/2
1	70.70
1	83.23
2	69.58
3	95.20
4	70.86
5	84.98
6	80.43
6	100.00
7	100.00
7	100.00
7	100.00
7	100.00
8	63.30
9	72.67
10	96.64
11	79.14
12	79.42

UNCLASSIFIED

13	73.76
14	87.37
15	92.75
16	81.95
16	93.06
16	89.14
16	84.63
16	93.66
16	86.29
17	79.04
18	82.19
19	88.63
20	83.65
21	72.65
22	72.78
22	87.36
22	86.40
23	88.93
24	91.43
25	86.37

37 records selected.

We selected every record since we are comparing the column on the current record to the same column on every record in the table and of course it will be greater than or equal to at least one other record in the table.

**Example 4.17.10**

List the average grades for all classes taken by any student taking more than two classes and where the student's average grade is at least as high as the overall student average for students taking at least three classes.

```
select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
  from CLASS
    where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 >=
( select avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
  from CLASS
 where CLASS_STUDENT = any
  ( select CLASS_STUDENT
    from CLASS
   group by CLASS_STUDENT
  having count (*) > 2 ) )
   and CLASS_STUDENT = any
   ( select CLASS_STUDENT
     from CLASS
    group by CLASS_STUDENT
   having count (*) > 2 ) ;
```

# UNCLASSIFIED

CLASS_STUDENT	(CLASS_SEM_1+CLASS_SEM_2)/2
7	100.00
7	100.00
7	100.00
7	100.00
16	93.66
16	93.06

6 records selected.

You can also use the operators "in" and "not in" with a subquery. Use "in" when you wish to select records which match a record in the list of selected records from the subquery. Use "not in" when you wish to select records which do not match any record in the list of selected records from the subquery.

## Example 4.17.11

For example list any student and his average grade from a list of students who are taking more than two classes.

```
select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
  from CLASS
  where CLASS_STUDENT in
( select CLASS_STUDENT
  from CLASS
  group by CLASS_STUDENT
  having count (*) > 2 ) ;
```

CLASS_STUDENT	(CLASS_SEM_1+CLASS_SEM_2)/2
7	100.00
7	100.00
7	100.00
7	100.00
16	81.95
16	93.66
16	86.29
16	84.63
16	89.14
16	93.06
22	72.78
22	86.40
22	87.36

13 records selected.

## Example 4.17.12

We could also list any student and his average grade from a list of students who are not taking fewer than two classes.

UNCLASSIFIED

```
select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
from CLASS
where CLASS_STUDENT not in
( select CLASS_STUDENT
  from CLASS
  group by CLASS_STUDENT
  having count (*) <= 2 ) ;
```

CLASS_STUDENT	(CLASS_SEM_1+CLASS_SEM_2)/2
7	100.00
7	100.00
7	100.00
7	100.00
16	81.95
16	93.06
16	89.14
16	84.63
16	93.66
16	86.29
22	72.78
22	87.36
22	86.40

13 records selected.

#### 4.18 HAVING

The group by clause allows you to group your records together based on a common element. A where clause is used only to select or reject individual records. It cannot be used to select or reject entire groups of records. A where clause cannot use aggregate functions for comparison since they relate to entire groups of records and not individual records. The having clause allows you to select or reject an entire group formed by the group by clause and to use aggregate functions for comparison. The having clause must always be used with a group by clause. A query may contain both a where clause and a having clause in which case the where clause is used to select the individual records which will make up the groups and the having clause is used to select the groups. A having clause may contain a nested query. The format of the having clause is:

```
having AGGREGATE_FUNCTION OPERATOR
      CONSTANT or AGGREGATE_FUNCTION or NESTED_QUERY
```

##### Example 4.18.1

List the departments having more than ten class hours.

```
select COURSE_DEPT, sum ( COURSE_HOURS)
  from COURSE
   group by COURSE_DEPT
   having sum ( COURSE_HOURS ) > 10 ;
```

COURSE_DEPT	SUM(COURSE_HOURS)
2	17
3	12

##### Example 4.18.2

List all the departments with more than three classes.

```
select COURSE_DEPT, count (*)
  from COURSE
   group by COURSE_DEPT
   having count (*) > 3 ;
```

COURSE_DEPT	COUNT(*)
2	4

##### Example 4.18.3

List the number of students from Virginia, District of Columbia, Maryland, North Carolina and Pennsylvania grouping them by their majors within their home states, but list only those groups having an average number of student years greater than two.

```
select ST_STATE, ST_MAJOR, count (*)
```

# UNCLASSIFIED

```

from STUDENT
  where ST_STATE in ( 'VA', 'DC', 'MD', 'NC', 'PA' )
    group by ST_STATE, ST_MAJOR
      having avg ( ST_YEAR ) > 2 ;

```

ST	ST_MAJOR	COUNT(*)
DC	3	1
MD	1	1
MD	4	2
MD	5	1
NC	1	1
PA	3	1
PA	4	2
PA	5	1
VA	1	1
VA	3	2

10 records selected.

## Example 4.18.4

List classes taken in department two and four in which the student grades were above the average for all classes.

```

select CLASS_COURSE, avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
  from CLASS
    where CLASS_DEPT = 2 or CLASS_DEPT = 4
      group by CLASS_COURSE
        having avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 ) >
          ( select avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
            from CLASS ) ;

```

CLASS_COURSE	AVG((CLASS_SEM_1+CLASS_SEM_2)/2)
201	88.00
202	92.75
401	86.06
402	96.31
403	96.29

## Example 4.18.5

List the average semester grades for the classes which have the students with the highest and the lowest average class grade.

```

select CLASS_COURSE, avg ( CLASS_SEM_1 ), avg ( CLASS_SEM_2 )
  from CLASS
    group by CLASS_COURSE
      having CLASS_COURSE =
        ( select CLASS_COURSE
          from CLASS

```



# UNCLASSIFIED

```

where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
  ( select max ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
    from CLASS )
or ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
  ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
    from CLASS ) ;

```

ERROR: ORA-1427: single-row subquery returns more than one row

no records selected

Oops! We got an error because the highest and/or lowest student grades are shared by more than one student. We will have to use the "any" qualifier for the subquery. Let's try again.

## Example 4.18.6

This time list the average semester grades for the classes which have the students with the highest and the lowest average class grade and remember to use the "any" qualifier.

```

select CLASS_COURSE, avg ( CLASS_SEM_1 ), avg ( CLASS_SEM_2 )
from CLASS
group by CLASS_COURSE
having CLASS_COURSE = any
  ( select CLASS_COURSE
    from CLASS
    where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
      ( select max ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
        from CLASS )
    or ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
      ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
        from CLASS ) ) ;

```

CLASS_COURSE	AVG(CLASS_SEM_1)	AVG(CLASS_SEM_2)
401	92.23	79.88
402	98.78	93.84
403	94.28	98.29
502	68.52	68.54
503	90.63	87.37

## Example 4.18.7

List the students and their second semester grades for the class which has the highest average grade of all students in the second semester.

```

select CLASS_STUDENT, CLASS_SEM_2
from CLASS
where CLASS_COURSE =
  ( select CLASS_COURSE
    from CLASS

```

# UNCLASSIFIED

```
group by CLASS_COURSE
having avg ( CLASS_SEM_2 ) =
( select max ( avg ( CLASS_SEM_2 ) )
from CLASS
group by CLASS_COURSE ) ) ;
```

CLASS_STUDENT	CLASS_SEM_2
3	97.48
7	100.00
16	97.40

## 4.19 Joining Multiple Tables

So far we've only been able to select data from one table in each query. Frequently you will wish to use the data from two or more tables in the same query. For example when we have listed the student while selecting from the class table we get the student's id number, not his name. Normally we'd want to list the name from the student table and the grading information from the class table. We would do this by listing more than one table in the from clause. This is called joining tables. The format of a query joining tables is:

```
select COLUMN, COLUMN, ...
from TABLE, TABLE, ...
... ..
```

All clauses that may be used with a select, such as where, order by, group by, having and nested queries, may also be used when joining tables. These clauses will be used to specify how the tables should be joined together. Generally you will specify the join in the where clause by having a statement that links columns from two tables together. If you do not specify how to join the tables you will get a list of every entry in each table joined together.

### Example 4.19.1

For example list all columns in the department and the professor table in a joined query.

```
select *
from DEPARTMENT, PROFESSOR ;
```

DEPT_ID	DEPT_DES	PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	History	1	Dysart	Gregory	3	3	35000.00
2	Math	1	Dysart	Gregory	3	3	35000.00
3	Science	1	Dysart	Gregory	3	3	35000.00
4	Language	1	Dysart	Gregory	3	3	35000.00
5	Art	1	Dysart	Gregory	3	3	35000.00
1	History	2	Hall	Elizabeth	4	7	45000.00
2	Math	2	Hall	Elizabeth	4	7	45000.00
3	Science	2	Hall	Elizabeth	4	7	45000.00
4	Language	2	Hall	Elizabeth	4	7	45000.00
5	Art	2	Hall	Elizabeth	4	7	45000.00
1	History	3	Steinbacner	Moris	2	1	30000.00
2	Math	3	Steinbacner	Moris	2	1	30000.00

# UNCLASSIFIED

3 Science	3 Steinbacner	Moris	2	1	30000.00
4 Language	3 Steinbacner	Moris	2	1	30000.00
5 Art	3 Steinbacner	Moris	2	1	30000.00
1 History	4 Bailey	Bruce	5	15	50000.00
2 Math	4 Bailey	Bruce	5	15	50000.00
3 Science	4 Bailey	Bruce	5	15	50000.00
4 Language	4 Bailey	Bruce	5	15	50000.00
5 Art	4 Bailey	Bruce	5	15	50000.00
1 History	5 Clements	Carol	1	4	40000.00
2 Math	5 Clements	Carol	1	4	40000.00
3 Science	5 Clements	Carol	1	4	40000.00
4 Language	5 Clements	Carol	1	4	40000.00
5 Art	5 Clements	Carol	1	4	40000.00

25 records selected.

You can see how every entry in the department table is joined with each entry in the professor table. Also note how this query produced more data than can fit across one line and wrapped the information for each record onto two lines. Exactly how this is done is up to your DBMS. Joining output in this way is really quite useless. However joins are very useful when used correctly. The professor table lists a department for each professor. But only the department id is included in the professor table, so by looking at records selected from the professor table we see a number in the column for department. This is pretty useless unless we then cross reference the department id in the department table to find the description of the department assigned to the professor. Remember why we used the id in the professor record instead of the complete description. It was to minimize the information stored in each table and to allow us to point to the detailed information.

## Example 4.19.2

Let's join the department and professor table to produce a list of the professors and a description of the department assigned to them. We will do this by joining on the prof\_dept column of the professor table and the dept\_id column of the department table.

```
select PROF_FIRST, PROF_NAME, DEPT_DESC
from PROFESSOR, DEPARTMENT
where PROF_DEPT = DEPT_ID ;
```

PROF_FIRST	PROF_NAME	DEPT_DESC
Carol	Clements	History
Moris	Steinbacner	Math
Gregory	Dysart	Science
Elizabeth	Hall	Language
Bruce	Bailey	Art

## Example 4.19.3

List the description of the department and the course and the professor's first and last name for each course offered.

UNCLASSIFIED

```
select DEPT_DESC, COURSE_DESC, PROF_FIRST, PROF_NAME
from PROFESSOR, DEPARTMENT, COURSE
where COURSE_DEPT = DEPT_ID
and COURSE_PROF = PROF_ID ;
```

DEPT_DES	COURSE_DESC	PROF_FIRST	PROF_NAME
Science	Chemistry	Gregory	Dysart
Science	Biology	Gregory	Dysart
Science	Physics	Gregory	Dysart
Language	French	Elizabeth	Hall
Language	Russian	Elizabeth	Hall
Math	Algebra	Moris	Steinbacner
Math	Calculus	Moris	Steinbacner
Math	Trigonometry	Moris	Steinbacner
Math	Geometry	Moris	Steinbacner
Art	Sculpture	Bruce	Bailey
Art	Music	Bruce	Bailey
History	World History	Carol	Clements
Art	Dance	Carol	Clements
History	Political History	Carol	Clements
Language	Spanish	Carol	Clements
History	Ancient History	Carol	Clements

16 records selected.

#### Example 4.19.4

Again list the description of the department and the course and the professor's first and last name for each course offered but this time order it by the department id and the course id.

```
select DEPT_DESC, COURSE_DESC, PROF_FIRST, PROF_NAME
from PROFESSOR, DEPARTMENT, COURSE
where COURSE_DEPT = DEPT_ID
and COURSE_PROF = PROF_ID
order by DEPT_ID, COURSE_ID ;
```

DEPT_DES	COURSE_DESC	PROF_FIRST	PROF_NAME
History	World History	Carol	Clements
History	Political History	Carol	Clements
History	Ancient History	Carol	Clements
Math	Algebra	Moris	Steinbacner
Math	Geometry	Moris	Steinbacner
Math	Trigonometry	Moris	Steinbacner
Math	Calculus	Moris	Steinbacner
Science	Chemistry	Gregory	Dysart
Science	Physics	Gregory	Dysart
Science	Biology	Gregory	Dysart
Language	French	Elizabeth	Hall
Language	Spanish	Carol	Clements
Language	Russian	Elizabeth	Hall

# UNCLASSIFIED

Art	Sculpture	Bruce	Bailey
Art	Music	Bruce	Bailey
Art	Dance	Carol	Clements

16 records selected.

## Example 4.19.5

And once again list the description of the department and the course and the professor's first and last name for each course offered but this time order it alphabetically by department and course.

```
select DEPT_DESC, COURSE_DESC, PROF_FIRST, PROF_NAME
from PROFESSOR, DEPARTMENT, COURSE
where COURSE_DEPT = DEPT_ID
and COURSE_PROF = PROF_ID
order by DEPT_DESC, COURSE_DESC ;
```

DEPT_DESC	COURSE_DESC	PROF_FIRST	PROF_NAME
Art	Dance	Carol	Clements
Art	Music	Bruce	Bailey
Art	Sculpture	Bruce	Bailey
History	Ancient History	Carol	Clements
History	Political History	Carol	Clements
History	World History	Carol	Clements
Language	French	Elizabeth	Hall
Language	Russian	Elizabeth	Hall
Language	Spanish	Carol	Clements
Math	Algebra	Moris	Steinbacner
Math	Calculus	Moris	Steinbacner
Math	Geometry	Moris	Steinbacner
Math	Trigonometry	Moris	Steinbacner
Science	Biology	Gregory	Dysart
Science	Chemistry	Gregory	Dysart
Science	Physics	Gregory	Dysart

16 records selected.

## Example 4.19.6

List the department description, course description, professor's last name and student's last name for the class in which a student earned the highest first semester grade the lowest first semester grade, the highest second semester grade and the lowest semester grade. Be sure to list only one record per student/course/department/professor combination. Also sort the list by department, course, professor, student

```
select DEPT_DESC, COURSE_DESC, PROF_NAME, ST_NAME
from DEPARTMENT, COURSE, PROFESSOR, STUDENT, CLASS
where CLASS_STUDENT = any
( select CLASS_STUDENT
```

# UNCLASSIFIED

```

from CLASS
  where CLASS_SEM_1 =
    ( select max ( CLASS_SEM_1 )
      from CLASS )
  or   CLASS_SEM_1 =
    ( select min ( CLASS_SEM_1 )
      from CLASS )
  or   CLASS_SEM_2 =
    ( select max ( CLASS_SEM_2 )
      from CLASS )
  or   CLASS_SEM_2 =
    ( select min ( CLASS_SEM_2 )
      from CLASS ) )
and CLASS_STUDENT = ST_ID
and CLASS_DEPT = DEPT_ID
and CLASS_COURSE = COURSE_ID
and COURSE_PROF = PROF_ID
  group by ST_NAME, COURSE_DESC, DEPT_DESC, PROF_NAME
  order by DEPT_DESC, COURSE_DESC, PROF_NAME, ST_NAME ;

```

DEPT_DES	COURSE_DESC	PROF_NAME	ST_NAME
Art	Dance	Clements	Guiffre
History	Ancient History	Clements	McGinn
Language	French	Hall	Guiffre
Language	Russian	Hall	Guiffre
Language	Spanish	Clements	Guiffre
Language	Spanish	Clements	Hess
Science	Biology	Dysart	Horrigan
Science	Chemistry	Dysart	Hess
Science	Physics	Dysart	Horrigan

9 records selected.

You do not always have to use an equality to join tables together. You may use any comparison operator.

## Example 4.19.7

For example select the professor's name, number of year's employed, salary and the minimum and maximum suggested salary for the number of years employed.

```

select PROF_NAME, PROF_YEARS, SAL_YEAR, SAL_END, PROF_SALARY,
      SAL_MIN, SAL_MAX
from PROFESSOR, SALARY
  where PROF_YEARS >= SAL_YEAR
    and PROF_YEARS <= SAL_END ;

```

PROF_NAME	PROF_YEARS	SAL_YEAR	SAL_END	PROF_SALARY	SAL_MIN	SAL_MAX
Steinbacner	1	1	1	30000.00	20000.00	29999.00

# UNCLASSIFIED

Dysart	3	3	3	35000.00	35000.00	39999.00
Clements	4	4	4	40000.00	40000.00	44999.00
Hall	7	6	10	45000.00	50000.00	51999.00
Bailey	15	11	15	50000.00	52000.00	53999.00

## Example 4.19.8

List the professor's name, salary, number of years employed, the year range for the suggested salary and the suggested salary range which the professor's salary falls into.

```
select PROF_NAME, PROF_SALARY, PROF_YEARS, SAL_YEAR, SAL_END,
       SAL_MIN, SAL_MAX
from PROFESSOR, SALARY
where PROF_SALARY between SAL_MIN and SAL_MAX ;
```

PROF_NAME	PROF_SALARY	PROF_YEARS	SAL_YEAR	SAL_END	SAL_MIN	SAL_MA
Steinbacner	30000.00	1	2	2	30000.00	34999.00
Dysart	35000.00	3	3	3	35000.00	39999.00
Clements	40000.00	4	4	4	40000.00	44999.00
Hall	45000.00	7	5	5	45000.00	49999.00
Bailey	50000.00	15	6	10	50000.00	51999.00

## 4.20 Correlation Names

There are times when you will have to specify a table name for a column to make it clear which column you want. It is possible to have columns in different tables with the same name. We avoided duplicate column names when we set up our tables. There are two ways to specify which table a column is from. The first is to prefix the column name with the table name in the format of:

TABLE\_NAME.COLUMN\_NAME

The second is through use of a correlation name. With this method you assign a name to the table in the from clause and use this name as a prefix when ever referencing the column. The format of correlation assignment is:

```
select CORRELATION_NAME.COLUMN_NAME ...
from TABLE_NAME CORRELATION_NAME, ...
```

### Example 4.20.1

Select department description, course description, professor's name and student's name for the student(s) taking four or more classes. Qualify all column names with the table names. This is a nested query, only qualify columns in the outer query.

```
select DEPARTMENT.DEPT_DESC, COURSE.COURSE_DESC, PROFESSOR.PROF_NAME,
       STUDENT.ST_NAME
from DEPARTMENT, COURSE, PROFESSOR, STUDENT, CLASS
where STUFNT.ST_ID = any
( select CLASS_STUDENT
```

# UNCLASSIFIED

```

from CLASS
  group by CLASS_STUDENT
    having COUNT (*) >= 4 )
and CLASS.CLASS_STUDENT = STUDENT.ST_ID
and CLASS.CLASS_COURSE = COURSE.COURSE_ID
and COURSE.COURSE_DEPT = DEPARTMENT.DEPT_ID
and COURSE.COURSE_PROF = PROFESSOR.PROF_ID ;

```

DEPT_DES	COURSE_DESC	PROF_NAME	ST_NAME
Language	French	Hall	Guiffre
Language	Russian	Hall	Guiffre
Art	Dance	Clements	Guiffre
Language	Spanish	Clements	Guiffre
History	World History	Clements	Williams
Science	Physics	Dysart	Williams
Language	Russian	Hall	Williams
Art	Sculpture	Bailey	Williams
History	World History	Clements	Williams
Math	Calculus	Steinbacner	Williams

10 records selected.

## Example 4.20.2

Do the same selection as in the previous example but this time assign correlation names to the tables and use them to qualify the columns.

```

select D.DEPT_DESC, C.COURSE_DESC, P.PROF_NAME, S.ST_NAME
from DEPARTMENT D, COURSE C, PROFESSOR P, STUDENT S, CLASS CL
where S.ST_ID = any
  ( select CLASS_STUDENT
    from CLASS
      group by CLASS_STUDENT
        having COUNT (*) >= 4 )
and CL.CLASS_STUDENT = S.ST_ID
and CL.CLASS_COURSE = C.COURSE_ID
and C.COURSE_DEPT = D.DEPT_ID
and C.COURSE_PROF = P.PROF_ID ;

```

DEPT_DES	COURSE_DESC	PROF_NAME	ST_NAME
Language	French	Hall	Guiffre
Language	Russian	Hall	Guiffre
Art	Dance	Clements	Guiffre
Language	Spanish	Clements	Guiffre
History	World History	Clements	Williams
Science	Physics	Dysart	Williams
Language	Russian	Hall	Williams
Art	Sculpture	Bailey	Williams
History	World History	Clements	Williams
Math	Calculus	Steinbacner	Williams



## UNCLASSIFIED

10 records selected.

### 4.21 Self Joins

There will be times when you wish to join the same table together as two or more tables. To do this you must use correlation names for the tables and then qualify the column names.

#### Example 4.21.1

List the names and salaries of the professors earning the same amount or more than Professor Hall.

```
select X.PROF_NAME, X.PROF_SALARY
from PROFESSOR X, PROFESSOR Y
where X.PROF_SALARY >= Y.PROF_SALARY
and Y.PROF_NAME = 'Hall' ;
```

PROF_NAME	PROF_SALARY
Hall	45000.00
Bailey	50000.00

You only need to assign correlation names to tables where confusion might arise. You may select from several tables where only some of them have correlation names.

#### Example 4.21.2

List the names, salaries, department and courses taught for the professors earning the same amount or more than Professor Hall.

```
select X.PROF_FIRST, X.PROF_NAME, X.PROF_SALARY, DEPT_DESC, COURSE_DESC
from PROFESSOR X, PROFESSOR Y, DEPARTMENT, COURSE
where X.PROF_SALARY >= Y.PROF_SALARY
and Y.PROF_NAME = 'Hall'
and X.PROF_ID = COURSE_PROF
and COURSE_DEPT = DEPT_ID ;
```

PROF_FIRST	PROF_NAME	PROF_SALARY	DEPT_DESC	COURSE_DESC
Elizabeth	Hall	45000.00	Language	French
Elizabeth	Hall	45000.00	Language	Russian
Bruce	Bailey	50000.00	Art	Sculpture
Bruce	Bailey	50000.00	Art	Music

### 4.22 EXISTS

Exists is a logical operator which is used in the where clause before a subquery. It will return a true if

# UNCLASSIFIED

the subquery returns at least one record and a false if it returns no records. The record being studied in the outer query is selected or rejected based on the true or false status of the subquery following the exists operator. The subquery associated with the exists operator must refer to at least one column in the outer query if results are to be correct. If no column is referred to in the outer query then all queries will be selected or rejected based on the result of the subquery which will always be the same.

## Example 4.22.1

List all professors who earn more than \$40000.00. Use the exists operator to perform this query. This is not a good example of when the exists operator should be used. It is to show the importance of referencing a column in the outer query.

```
select *
  from PROFESSOR X
   where exists
    ( select *
      from PROFESSOR
      where X.PROF_SALARY > 40000.00 ) ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
2	Hall	Elizabeth	4	7	45000.00
4	Bailey	Bruce	5	15	50000.00

Note how we use a correlation name for the professor table in the outer query and then used that table for the prof\_salary column in the inner query. This query says look at every record in the professor table and for each record if the prof\_salary column is greater than \$40000.00 return true and select this record.

## Example 4.22.2

Now do the same query without the correlation name.

```
select *
  from PROFESSOR
   where exists
    ( select *
      from PROFESSOR
      where PROF_SALARY > 40000.00 ) ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

Note how in this case the subquery returns true if one or more record is selected. Without a reference to a column in the outer query the result of the subquery will always be the same. Since in this case

UNCLASSIFIED

the subquery is true all records are selected for the outer query.

**Example 4.22.3**

I want to know the average salary earned by professors teaching one or more courses with more than three semester hours. Be sure to count a professor's salary only one regardless of the number of qualifying courses he teaches.

```
select avg (PROF_SALARY)
  from PROFESSOR
  where exists
    ( select *
      from COURSE
      where COURSE_HOURS > 3
        and PROF_ID = COURSE_PROF ) ;
```

```
AVG(PROF_SALARY)
36666.67
```

Note that I did not use a correlation name for the table in the outer query. It was not necessary in this case since any column from the table professor could only come from the outer query since the inner query is selecting only from the table course. This query must use the exists operator in order to get the correct results. We counted each professor only once regardless of how many courses of over three credit hours he teaches.

**Example 4.22.4**

List the salary and professor id from all the records selected to form the average in the previous example.

```
select PROF_SALARY, PROF_ID
  from PROFESSOR
  where exists
    ( select *
      from COURSE
      where COURSE_HOURS > 3
        and PROF_ID = COURSE_PROF ) ;
```

PROF_SALARY	PROF_ID
35000.00	1
45000.00	2
30000.00	3

We selected three professors, no duplicates.

**Example 4.22.5**

## UNCLASSIFIED

Now do the same query without the exists operator. Simply select the average salary for professors where the professor's id matches the teacher for a course from the course table and that course is more than three semester hours.

```
select avg (PROF_SALARY)
  from PROFESSOR, COURSE
   where COURSE_HOURS > 3
     and PROF_ID = COURSE_PROF ;
```

```
AVG(PROF_SALARY)
      33571.43
```

And we come up with a different average. How can this be? Because we counted the salary for each professor for each course they teach. Some of the professors must have been counted more than once to arrive at this average.

### Example 4.22.6

List the records which were used to arrive at the average in the above example.

```
select PROF_SALARY, PROF_ID, COURSE_HOURS, COURSE_ID
  from PROFESSOR, COURSE
   where COURSE_HOURS > 3
     and PROF_ID = COURSE_PROF ;
```

PROF_SALARY	PROF_ID	COURSE_HOURS	COURSE_ID
35000.00	1	5	302
35000.00	1	4	303
45000.00	2	4	403
30000.00	3	4	201
30000.00	3	5	203
30000.00	3	4	204
30000.00	3	4	202

7 records selected

As you can see professor one and three were selected multiple times since they teach several courses of over three semester hours.

### 4.23 INSERT INTO

The "insert into" command is used to add new records to a table. The values added as columns of a record may be literals or values returned as the result of a subquery. In section 4.2 we inserted literals into the columns of our tables. The format of the literal "insert into" statement is:

```
insert into TABLE
values
( COLUMN_1_DATA, COLUMN_2_DATA, ... ) ;
```

You must supply data for every column in the table. Character strings must be enclosed in single quotes. Character string columns should be the maximum full length of the column. When a character string field won't fill up the column it is advisable that you pad it with spaces. The unused characters in a character string must be ascii spaces when using Ada/SQL. Otherwise you may end up with a data type incompatibility when accessing the field in an Ada/SQL program. Some DBMSs will automatically pad with spaces. Others will pad with a null value which is not ascii spaces. If you are not sure how your DBMS will pad a character string fill it with spaces yourself. Likewise with numeric fields, pick your own "null" value and stick to it. The null value assigned by the DBMS may not be compatible with Ada/SQL. In our examples we use zeros to pad numeric fields.

#### Example 4.23.1

Add a new record to the STUDENT table. This will be student number 26, Samuel Brenner, from California majoring in Art. This is his first year. We'll put him in dorm room A101.

```
insert into STUDENT
values
( 026, 'Brenner      ', 'Samuel      ', 'A101', 'CA', 5, 1 ) ;
```

1 record created.

If you do not want to insert a value into each column of the record you may specify which columns you are supplying data for. All remaining columns will contain the null value designated by your DBMS. This may cause problems with data type compatibility in Ada/SQL. Before using this method be sure you know what your DBMS null values are. When specifying the columns to fill with data in an "insert into" statement the format is:

```
insert into TABLE
( COLUMN_1, COLUMN_2, ... )
values
( COLUMN_1_DATA, COLUMN_2_DATA, ... ) ;
```

You do not have to enter the columns in the order in which they appear in the table. But you must enter the column names in the same order as the literals to be inserted into the columns.

Remember, when inserting data into a column which is designated as "not\_null" you will get an error if the insert value is null. Also, when a column is designated as "unique" you will get an error if the data being inserted into that column is a duplicate of the data in that column of another record.

# UNCLASSIFIED

## Example 4.23.2

Add a student who we don't know much about. The only information is a last name of Mamout, from Alaska and this is the first year of study. Don't assign a student id number yet. We'll do that later on when we know more about this student.

```
insert into STUDENT
( ST_YEAR, ST_STATE, ST_NAME )
values
( 1, 'AK', 'Mamout      ' ) ;
```

1 record created.

## Example 4.23.3

Before inserting the last two students in our table we had students with id numbers between 1 and 25. Let's list all students who's id falls outside that range.

```
select *
from STUDENT
where ST_ID not between 1 and 25 ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
26	Brenner	Samuel	A101	CA	5	1

Our DBMS did not list Mamout who has a null student id number. Some DBMSs may list that record here. Ours will not consider a null value for comparisons.

## Example 4.23.4

Let's pull up Mamout's record by selecting all student who's name starts with an M.

```
select *
from STUDENT
where ST_NAME like ('M%') ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
2	McGinn	Gregory	A102	MD	1	3
5	McNamara	Howard	A201	VA	5	1
20	McMurray	Eric	B204	VA	2	1
22	Martin	Charoltte	C102	DC	1	2
24	Martin	Edward	C104	MD	5	3
	Mamout			AK		1

6 records selected.

And there is Mamout with only the information we supplied.

## UNCLASSIFIED

Instead of supplying literals for the data to be inserted into columns the results of a subquery can be used. All the records selected by the subquery will be inserted into the table. The subquery "select" takes the place of the "values" clause. If you don't list any column names all columns of the table will be filled. If you specify column names to be filled then only those columns will contain data. The format of the subquery "insert into" statement where all columns are to be filled is:

```
insert into TABLE
  select SELECT_COLUMN_1, SELECT_COLUMN_2, ...
  from TABLE
  remaining clauses in the subquery ;
```

The columns listed in the select clause, SELECT\_COLUMN\_1, etc., must be in the same order and of compatible data types as the columns listed in the table in the insert clause.

The format of the subquery "insert into" statement where only selected columns are to be filled is:

```
insert into TABLE ( COLUMN_1, COLUMN_2, ... )
  select SELECT_COLUMN_1, SELECT_COLUMN_2, ...
  from TABLE
  remaining clauses in the subquery ;
```

The columns listed in the select clause, SELECT\_COLUMN\_1, etc., must be in the same order and of compatible data types as the specified columns in the table clause of the insert clause.

### Example 4.23.5

Remember our GRADES table which we have left empty so far? Let's check it to see if it's still empty.

```
select *
  from GRADE ;

no records selected
```

### Example 4.23.6

Let's insert into the GRADE table all classes taught in department 5 and the average grade earned in those classes.

```
insert into GRADE
  select CLASS_COURSE, avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
  from CLASS
  where CLASS_DEPT = 5
  group by CLASS_COURSE ;

3 records created.
```

### Example 4.23.7

List out the contents of the GRADE table.

## UNCLASSIFIED

```
select *  
from GRADE ;
```

GRADE_COURSE	GRADE_AVERAGE
501	82.86
502	68.53
503	89.00

### Example 4.23.8

Insert into the GRADE table the average grade for all classes, leave the course column of the table empty.

```
insert into GRADE ( GRADE_AVERAGE )  
select avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )  
from CLASS ;
```

1 record created.

### Example 4.23.9

And list out the information in the GRADE table now.

```
select *  
from GRADE ;
```

GRADE_COURSE	GRADE_AVERAGE
501	82.86
502	68.53
503	89.00
	85.08

## 4.24 UPDATE

When you want to modify columns in a record in a table you will use the update statement. This allows you to change one or more columns in one or more records of a table. Columns may be modified with literal values, query statements or expressions. A where clause specifies the record(s) to be changed. If no where clause is used, all records in the table will be changed. The format of the update statement is:

```
update TABLE  
set COLUMN_1 = value ,  
    COLUMN_2 = value , ...  
where expression ;
```

Value in the "set" clause is either a literal value, a query returning a value for the column or an expression. If value is a query it must be enclosed in parenthesis and return the same number of columns and



UNCLASSIFIED

of the same data type as the columns specified in the set clause. Expression in the "where" clause is any expression valid in a where clause and determines which records will be modified.

**Example 4.24.1**

Remember the student, Mamout, from Alaska that we had just added. Let's take a look at his record again.

```
select *
  from STUDENT
   where ST_NAME = 'Mamout' ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
	Mamout	AK		1		

**Example 4.24.2**

We now want to fill in the empty columns in his record. His id will be 27, his first name is Mark, room number B101 and his major is Science. Let's update his record with this information.

```
update STUDENT
  set ST_ID = 27,
      ST_FIRST = 'Mark',
      ST_ROOM = 'B101',
      ST_MAJOR = 3
   where ST_NAME = 'Mamout' ;
```

1 record updated.

This update statement uses literals to update all the columns which we wish to modify.

**Example 4.24.3**

Display Mark Mamout's record with the updates.

```
select *
  from STUDENT
   where ST_NAME = 'Mamout' ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
27	Mamout	Mark	B101	AK	3	1

**Example 4.24.4**

List all records in the professor table.

```
select *
```

UNCLASSIFIED

from PROFESSOR ;

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

**Example 4.24.5**

We will now give a 5% raise to all professors who have been with the school for more than 10 years.

```
update PROFESSOR
  set PPOF_SALARY = PROF_SALARY * 1.05
    where PROF_YEARS > 10 ;
```

1 record updated.

This update is done with an expression where the new salary will be equal to the old salary multiplied by 1.05 which results in a 5% raise.

**Example 4.24.6**

Now let's look at all records which we updated in the above query.

```
select *
  from PROFESSOR
    where PROF_YEARS > 10 ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
4	Bailey	Bruce	5	15	52500.00

**Example 4.24.7**

We want to adjust professor Steinbacner's salary to 5% more than the average of our professors who have been working here less than five years.

```
update PROFESSOR
  set PROF_SALARY =
    ( select ( avg ( PROF_SALARY ) * 1.05 )
      from PROFESSOR
        where PROF_YEARS < 5 )
    where PROF_NAME = 'Steinbacner ' ;
```

1 record updated.

UNCLASSIFIED

This update is done using a select to determine the contents of the modified columns.

**Example 4.24.8**

And display professor Steinbacner's updated record.

```
select *
from PROFESSOR
where PROF_NAME = 'Steinbacner ' ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
3	Steinbacner	Moris	2	1	36750.00

**Example 4.24.9**

Give all of our professors the suggested raise stored in the salary table based on the number of years they have been with us.

```
update PROFESSOR X
set PROF_SALARY =
( select ( PROF_SALARY + ( PROF_SALARY * SAL_RAISE ) )
from PROFESSOR, SALARY
where X.PROF_YEARS between SAL_YEAR and SAL_END
and X.PROF_NAME = PROF_NAME ) ;
```

5 records updated.

This update is done with a query selection. Note how correlation names were necessary to link the columns in the update statement to those in the query statement.

**Example 4.24.10**

And list out the new information in the professor table.

```
select *
from PROFESSOR ;
```

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	36750.00
2	Hall	Elizabeth	4	7	45900.00
3	Steinbacner	Moris	2	1	37117.50
4	Bailey	Bruce	5	15	53550.00
5	Clements	Carol	1	4	41400.00

## 4.25 DELETE

The "delete" statement is used to remove one or more records from a table. A "where" clause specifies which record(s) are to be deleted. If the "where" clause is omitted then all records in the table are deleted. The "where" clause may include any expressions valid in a "where" clause including subqueries. The format of the "delete" statement is:

```
delete TABLE
  where expression ;
```

### Example 4.25.1

We will delete the record for the student named Bennett but first list that student's record.

```
select *
  from STUDENT
  where ST_NAME = 'Bennett' ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
11	Bennett	Nellie	A303	PA	4	3

### Example 4.25.2

Now delete Bennett's student record.

```
delete STUDENT
  where ST_NAME = 'Bennett' ;
```

1 record deleted.

### Example 4.25.3

We now want to delete the student record for Martin, list the record first.

```
select *
  from STUDENT
  where ST_NAME = 'Martin' ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
22	Martin	Charlotte	C102	DC	1	2
24	Martin	Edward	C104	MD	5	3

We have two Martins, Edward is the one we wish to delete.

### Example 4.25.4

# UNCLASSIFIED

Delete the record from the student table for Edward Martin.

```
delete STUDENT
  where ST_NAME =      'Martin      '
     and ST_FIRST =    'Edward      ' ;
```

1 record deleted.

## Example 4.25.5

Now list all records remaining in the student table for Bennett and Martin.

```
select *
  from STUDENT
   where ST_NAME = 'Bennett      '
     or ST_NAME = 'Martin       ' ;
```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
22	Martin	Charoltte	C102	DC	1	2

Charoltte Martin is the only one left since Edward Martin's and Nellie Bennett's records were deleted.

## Example 4.25.6

We now want to delete from the student table and the class table all records for the student who has the lowest average class grade. First list that student's id, the course and the average grade.

```
select CLASS_STUDENT, CLASS_COURSE, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
  from CLASS
   where CLASS_STUDENT =
         ( select CLASS_STUDENT
           from CLASS
          where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
                ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
                  from CLASS ) ) ;
```

CLASS_STUDENT	CLASS_COURSE	(CLASS_SEM_1+CLASS_SEM_2)/2
8	502	63.30

## Example 4.25.7

Now select all the information in the student table about this person. Use a nested query, not a "where" clause based on information gathered from the previous query.

```
select *
  from STUDENT
   where ST_ID =
         ( select CLASS_STUDENT
```

# UNCLASSIFIED

```

from CLASS
  where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
    ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
      from CLASS ) ) ;

```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
8	Hagan	Carl	A204	PA	5	4

## Example 4.25.8

We will now delete this student from the student table. Structure the delete statement to delete the student with the lowest class average, do not use information gathered in previous queries.

```

delete STUDENT
  where ST_ID = ( select CLASS_STUDENT
                  from CLASS
                    where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
                      ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
                        from CLASS ) ) ;

```

1 record deleted.

## Example 4.25.9

Now delete information about this student from the class table. Structure the delete statement to delete the student class information with the lowest class average, do not use information gathered in previous queries.

```

delete CLASS
  where CLASS_STUDENT =
    ( select CLASS_STUDENT
      from CLASS
        where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
          ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
            from CLASS ) ) ;

```

1 record deleted.

## Example 4.25.10

Now let's take a look at what's left in the student table.

```

select *
  from STUDENT ;

```

ST_ID	ST_NAME	ST_FIRST	ST_R	ST	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	4
2	McGinn	Gregory	A102	MD	1	3
3	Lewis	Molly	A103	PA	4	2

# UNCLASSIFIED

4	Waxler	Dennis	A104	NC	2	2
5	McNamara	Howard	A201	VA	5	1
6	Hess	Fay	A202	DC	3	3
7	Guiffre	Jennifer	A203	MD	4	1
9	Bearman	Rose	A301	VA	2	1
10	Thompson	Paul	A302	NC	1	3
12	Schmidt	John	A304	SC	5	2
13	Gevarter	Susan	B101	NY	5	4
14	Sherman	Donald	B102	VA	3	3
15	Gorham	Milton	B103	WV	2	2
16	Williams	Alvin	B104	DC	1	1
17	Woodliff	Dorothy	B201	MD	4	4
18	Ratliff	Ann	B202	NY	5	1
19	Phung	Kim	B203	SC	2	2
20	McMurray	Eric	B204	VA	2	1
21	O'Leary	Peggy	C101	PA	3	4
22	Martin	Charlotte	C102	DC	1	2
23	O'Day	Hilda	C103	NC	4	1
25	Chateauneuf	Chelsea	C105	VA	1	3
26	Brenner	Samuel	A101	CA	5	1
27	Mamout	Mark	B101	AK	3	1

24 records selected.

## Example 4.25.11

Delete all information in all tables now. Start by deleting the contents of the grade table.

```
delete GRADE ;
```

4 records deleted.

## Example 4.25.12

List the contents of the grade table.

```
select *
from GRADE ;
```

no records selected

## Example 4.25.13

Delete the contents of the department table.

```
delete DEPARTMENT;
```

5 records deleted.

UNCLASSIFIED

**Example 4.25.14**

Delete the contents of the professor table.

```
delete PROFESSOR;
```

5 records deleted.

**Example 4.25.15**

Delete the contents of the course table.

```
delete COURSE;
```

16 records deleted.

**Example 4.25.16**

Delete the contents of the student table.

```
delete STUDENT;
```

24 records deleted.

**Example 4.25.17**

Delete the contents of the class table.

```
delete CLASS;
```

36 records deleted.

**Example 4.25.18**

Delete the contents of the salary table.

```
delete SALARY;
```

9 records deleted.



## 5. Introduction To Programming With Ada/SQL

We are now going to learn how to write an Ada/SQL application program. First we'll write the general supporting modules necessary for an Ada/SQL program. Then as we study the different available Ada/SQL statements we will include the examples which we used in section 4 in our program. For the most part I will only have one query in the program at a time. This is so we can compile, link and run the program and see our results for each query before going on to the next. As you follow along save each query we program in another file. At the end we'll go back and run one large program containing all the queries.

Ada/SQL has two primary components, the Data Definition Language, DDL, which defines the tables we will be using and the Data Manipulation Language, DML, which are the actual query statements. There are several components of the Ada/SQL system which you will have to locate before attempting to use the system. If you are having problems locating some of the items I'll be discussing below, ask your database or systems person to assist you.

### 5.1 The Ada/SQL Library

In order to use the Ada/SQL system the Ada/SQL run time library will have to be in place on your system. This library contains all the workings to translate your Ada/SQL DML into something understandable by the underlying DBMS, to accept results from the DBMS and to return them to you in a standard way. At this time you should find out where your Ada/SQL library is located. You need not worry about the contents of this library. However you will not be able to compile your Ada/SQL application program without referencing the Ada/SQL library.

### 5.2 The Standard Modules

There are several files which must be located in a specified directory in order for the Ada/SQL system to run properly. These files are system and DBMS specific and their location must be defined to the Ada/SQL system. On some systems it may be hard coded into the Ada/SQL system, on others you may have to assign an environment. Find out at this time how this is done on your system and if necessary execute the command for correct assignment.

### 5.3 Your Sublibrary

The library into which you will be compiling your Ada/SQL application program must be a sublibrary of the Ada/SQL run time library mentioned above. This is because you will be making references to modules in that library. Also the modules created by the Application Scanner, which I'll get to in a minute, will use many of the modules in the run time library. So at this time create an empty library for your use, and make sure it's a sublibrary of the Ada/SQL run time library.

### 5.4 The Application Scanner

When you write an Ada/SQL application program many subroutines must be generated based on your data types, table structures and DML statements used. These routines are created by the Application Scanner. For each compilation unit you write which contains DML the Application Scanner

## UNCLASSIFIED

must be run generating a subroutine package. You will not be able to compile a compilation unit containing DML until you have compiled it's Application Scanner outputted generated function compilation unit. Every time you make a change to your compilation unit you will have to run the Application Scanner and recompile it's output also. You run the Application Scanner only with your compilation units containing DML. Any unit using DML will have references, "with" and "use" to units containing DDL. The DDL units will also be read and used by the Application Scanner. If a DDL unit is changed even if the DML unit is not you will still have to rescan the DML unit. You do not need to run the Application Scanner with compilation units which do not contain DML. Therefore it becomes good programming practice to code all DML in as few as possible compilation units. It is preferable to run the Application Scanner on one or two units than to have to run it one each and every unit in your program. Which is what you'd have to do if your DML statements were spread out throughout the program. It is, of course, possible to have DML in all compilation units, it simply will be more tedious to scan and compile all the scanner output.

You should find out at this time how to execute the Application Scanner on your system. You will enter a command such as "apscan" or "run apscan". Type in the command to begin execution now.

The Application Scanner will ask you several questions. Output from the Application Scanner is in darker print. The first two deal with the case of table and column names sent to the DBMS. Many DBMSs are insensitive to case and would consider table names of STUDENT and student to be identical. Some DBMSs are case sensitive and STUDENT and student would denote two different tables. As explained in a previous section you must take case into consideration when naming your tables and columns. All table names must be of the same case, and all column names must be of the same case. However table and columns need not be of the same case.

The first question asked by the Application Scanner is:

**Should table names be sent to the DBMS in upper case or lower case? Enter U for upper case (default) or L for lower case:**

You should respond with: U, u or carriage return if the names of the tables in the database are upper case. You should respond with: L or l if the names of the tables in the database are lower case.

The Application Scanner then asks:

**Should column names be sent to the DBMS in upper case or lower case? Enter U for upper case (default) or L for lower case:**

You should respond with: U, u or carriage return if the names of the columns in the tables are upper case. You should respond with: L or l if the names of the columns in the tables are lower case.

The Application Scanner then asks:

**Enter DML filename:**

This is where you will enter the name of your compilation unit which contains DML. For example here we will be using a unit named "test\_dml.ada". Enter the name followed by a carriage return.

The Application Scanner then asks:

**Enter listing filename:**

## UNCLASSIFIED

This is where you will enter the name of the listing file, the file which will contain any error messages put out by the Application Scanner. It is recommended that you use your compilation unit name, replacing the ".ada" extension with one that sounds like a listing file, such as ".lst". We will be using "test\_dml.lst". Enter the name followed by a carriage return.

The Application Scanner then asks:

**Enter filename for generated functions:**

This is where you will enter the name of the compilation unit which will be created by the Application Scanner. This unit will then become a part of your program and must be compiled into your library before your compilation unit from which it was generated. The package name used in the generated compilation unit will be that of the package or procedure name of the DML unit with an extension of "\_ADA\_SQL". For example for our DML unit names "TEST\_DML.ADA" the package name is "TEST\_DML\_EXAMPLES" and the package name of our generated package will be "TEST\_DML\_EXAMPLES\_ADA\_SQL". Your DML unit must "with" and "use" this generated package. It is recommended that you give a name to the generated package compilation unit which is descriptive, such as adding the extension "\_ADA\_SQL" to your compilation unit name. For example with the DML unit name of "TEST\_DML.ADA" use "TEST\_DML\_ADA\_SQL.ADA" for the generated package. Enter the name followed by a carriage return.

The Application Scanner then says:

**Invoking application scanner with:**

DML filename ==> TEST\_DML.ADA

Listing filename ==> TEST\_DML.LST

Generated package ==> TEST\_DML\_ADA\_SQL.ADA

The Application Scanner then thinks about things for a bit, checks your DDL and DML modules for accuracy and creates the generated package. If all went well the Application Scanner will respond with the message:

**%ADASQL-I-SCAN, Scan completed with no errors detected**

Your listing file will contain a listing of the unit scanned. Your generated function package will be ready for use. However if there was an error in your DML unit or in the DDL units which it references the Application Scanner will respond with the message:

**%ADASQL-I-SCAN, Scan completed with errors**

Your listing file will contain error messages which should be self-explanatory. Your generated function package will be useless, do not attempt to use one created when errors are encountered. Correct your errors and run the Application Scanner again.

### 5.5 Compiling Ada/SQL Programs

The compilation units of an Ada/SQL program are compiled as the units of any other Ada program would be. The library into which you compile must be a sublibrary of the Ada/SQL system library. The compilation units generated by the Application Scanner must be compiled before the DML unit from which it was generated.

## **5.6 Linking Ada/SQL Programs**

Ada/SQL programs are linked the same way any other Ada programs are linked. You may have to include libraries related to your DBMS in the link command. Check with your database or system person for more information.

## **5.7 Running Ada/SQL Programs**

An Ada/SQL program is executed just as any other program would be. Some systems require that some sort of start up is done with the DBMS before programs accessing it may run. Check with you database or system person about this.

## 6. The Units Making Up An Ada/SQL Program

There are several separate units which you will have to code in order to create an Ada/SQL application program. Your program must contain an authorization package, Data Definition Language which consists of data type definitions, table and column descriptions and descriptions of variables used to hold database values, and the Data Manipulation Language statements.

A schema is a group of compilation units containing an authorization identifier, the DDL and the DML necessary to perform a function.

Package names of the authorization package, data type definitions package, DDL package and variable packages must be the same as the name of the source file in which they are contained, minus the system extension indicating an Ada program, ".ADA" for example. For example a DDL package with a name of "DDL\_TEST\_PACKAGE" must be contained in a source file called "DDL\_TEST\_PACKAGE.ADA". This is done so that the text of the packages can be found by the Application Scanner when its name is encountered in a "with" clause.

In the format examples below for the various units. variables which you must fill in will be enclosed in angle brackets < >.

### 6.1 Elements Permitted In The DDL

Ada/SQL DDL may contain definitions only. It may contain no function specifications and call no subprograms. No package bodies are permitted in the DDL. Nested packages are not permitted except for the requires "ADA\_SQL" subpackage which you will see in examples below. Private sections, Ada attributes, renaming declarations, generic declarations, generic instantiations, deferred constant declarations, subprogram declarations, task declarations, exception declarations, object declarations and number declarations are not permitted. Constants and named numbers are not permitted. All ranges, index constraints etc. are to be defined with literals, not with variables, constants or complex expressions. All expressions are to be simple literals, no math may be performed, no functions such as ABS or NOT may be referenced, no relational operators such as = > or < may be used and no variables or constants may be used. Based literals are not permitted, all numeric literals must be decimal. No default values may be assigned to any types. Access types, private types, task types, incomplete type declarations and representation clauses may not be used. Array types must be made up of CHARACTER components and be of one dimension only with integer index. Records may not contain discriminants, variant parts or unconstrained arrays; records must be defined as being of fixed length. Record components may not be of record types. Fixed point numbers may not be used. Floating point and integer may be used. Type conversions may not be used.

Ada/SQL supports the predefined type CHARACTER which is an enumeration type with the values of the 128 characters of the ASCII code. Ada/SQL supports the predefined type BOOLEAN which is an enumeration type with the values of FALSE and TRUE. Ada/SQL supports the predefined integer type of INTEGER, as well as the NATURAL (zero and greater than zero) and POSITIVE (greater than zero) subtypes. Other optional predefined types defined in the STANDARD package of your system, such as SHORT\_INTEGER and LONG\_INTEGER are also supported. Ada/SQL supports the predefined real type FLOAT. Other optional predefined types defined in the STANDARD package of your system, such as SHORT\_FLOAT and LONG\_FLOAT are also supported. Ada/SQL supports the predefined type STRING. The values of the predefined type STRING are one-dimensional arrays of the predefined type CHARACTER, indexed by values of the predefined subtype POSITIVE. Any other

## UNCLASSIFIED

predefined types in your system package STANDARD and/or other system dependent packages may be used with Ada/SQL, provided they are within a class described above.

For string type definitions all arrays shall have a single integer index with components of CHARACTER type. For constrained array definitions, arrays shall have a single integer index with components of CHARACTER type, the index constraint shall have positive bounds, and the subtype of the components shall be of CHARACTER type and have no associated constraint. For unconstrained array definitions the index type shall be of an integer type. The components subtypes shall be of CHARACTER type and have no associated constraints.

Integer values are positive or negative integers or zero. Integer types may have range constraints which identify a lower and upper limit of valid numbers associated with the type.

Real or floating point numbers represent approximations of numbers, with precision to a specified number of significant digits, and an optional restriction on their range.

Enumeration data types may be used. Enumeration literals shall be identifiers or character literals. Each enumeration literal listed by an enumeration type definition must be unique. Each enumeration literal of an enumeration type has a position number which is an integer value, starting with zero. Enumeration literals are represented in the database by their position number.

Derived data types may be used, providing that the parent type conforms to the above rules.

### 6.2 Database Predefined Package

There is a predefined package called "DATABASE" which defines the following data types:

```
type INT          is new STANDARD.INTEGER;
type DOUBLE_PRECISION is new STANDARD.FLOAT;
type CHAR         is new STANDARD.STRING;
```

It may be "withed" and "used" from within Ada/SQL units.

### 6.3 Authorization Package'

Frequently a database system will allow groups of tables will be broken down into separate databases each containing several tables. Imagine that on the same computer system which we are storing the information about United University we also want to store information for an inventory for a hardware store and medical records for a doctor. It would be logical to split these into three databases. You would have to specify which database you wanted to use and you would not be allowed to share information across databases.

We have a feature in Ada/SQL to allow for multiple databases. The authorization package assigns an identifier to a database which is then used with all DDL for tables within that database. Multiple authorizations may not be mixed in the same schema. A program may contain more than one authorization identifier, set of DDL and DML, but any one compilation unit may not reference more than one authorization identifier.

The format of the authorization package is:

## UNCLASSIFIED

```
with SCHEMA_DEFINITION;  
use SCHEMA_DEFINITION;
```

```
package <authorization package name>  
    function <authorization identifier> is new AUTHORIZATION_IDENTIFIER;  
end <authorization package name> ;
```

Where:

SCHEMA\_DEFINITION is a package defined in the Ada/SQL library. You will see it used on several occasions. Where it is specified it must be included.

<authorization package name> is any valid Ada package name you desire. It is also required that the <authorization package name> be the same as the name of the source file in which it is contained.

The <authorization identifier> is any valid Ada identifier name. An <authorization identifier> may not duplicate another <authorization identifier> defined in the same Ada/SQL program.

Each authorization package must be contained in a separate source file.

### 6.4 Data Type Definition Package

All data types which will be used to define the columns of your tables and to define all variables used by the DML to accept data from the database must be defined in a data type definition package. The format of a data type definition package is:

<context clause>

package <package name> is

<use clause>

package ADA\_SQL is

<use clause>

    <type declarations>  
    <subtype declarations>  
    <derived type declarations>

end ADA\_SQL;

end <package name> ;

Where:

## UNCLASSIFIED

<context clause> may contain "with" and "use" statements to the predefined package "DATABASE" and to any other Ada/SQL data type definition package(s).

<package name> is any valid Ada package name and also must be the name of the source file.

<use clause> may "use" packages not permitted to appear in the "use" clause of the context clause such as the nested ADA\_SQL package of another data type definition package.

ADA\_SQL nested package is mandatory in all data type definition packages.

<type declarations> may define any data type valid for use with Ada/SQL, string, integer, floating point and enumeration. The identifier for a type declaration may not be the same as any table name or any other data type defined in the same schema. Type declaration identifiers may not contain the "\_NOT\_NULL" or "NOT\_NULL\_UNIQUE" suffix.

<subtype with Ada/SQL, string, integer, floating point and enumeration. The identifier for a subtype declaration may not be the same as any table name or any other data type defined in the same schema. If the last characters of the identifier of the subtype are "\_NOT\_NULL" or "NOT\_NULL\_UNIQUE" then the identifier of the subtype indicator must be identical to the name of the subtype identifier minus the suffix. The "\_NOT\_NULL" and "\_NOT\_NULL\_UNIQUE" suffixes are constraints placed upon data permitted in the database. Other constraints are defined using range constraints on type and subtype declarations.

<derived type declarations> may define a derived type of any data type valid for use with Ada/SQL, string, integer, floating point and enumeration. The identifier for a derived type declaration may not be the same as any table name or any other data type defined in the same schema. Derived type declaration identifiers may not contain the "\_NOT\_NULL" or "NOT\_NULL\_UNIQUE" suffix.

### 6.5 Table Definition Package

The table definition package is where the database tables to be used in this Ada/SQL program are defined. Each table which will be used must have its columns defined using the data types that were declared in the data type definition package. A table is defined as a record and the columns as elements of that record. The format of a table definition package is:

<context clause>

package <package name> is

<use clause>

package ADA\_SQL is

<use clause>



## UNCLASSIFIED

```
schema_authorization : identifier := <authorization identifier> ;
```

```
type <table name> is
  record
    <column definition>
    ...
  end record
```

```
end ADA_SQL;
```

```
end <package name> ;
```

Where:

<context clause> may contain "with" and "use" statements to the predefined package "DATABASE" and must contain "with" and "use" statements to the predefined package SCHEMA\_DEFINITION and to the authorization package whose identifier will be mentioned as the <authorization identifier> as well as any Ada/SQL data type definition packages having referenced types in this package.

<package name> is any valid Ada package name and also must be the name of the source file.

<use clause> may "use" packages not permitted to appear in the "use" clause of the context clause such as the nested ADA\_SQL package of the data type definition package.

<authorization identifier> is the identifier declared in the authorization package included in the context clause.

ADA\_SQL nested package is mandatory in all type definition packages.

<table name> is the name of the table exactly as it appears in the database. The table name may not be the same as the table name of any other table in this schema nor the same as any data type identifier.

<column definition> defines a column of the database table. Each column definition contains a column name which must match the name of the column exactly as it appears in the database and the indicated data type which must be of the same type as that defined in the database. Columns must be defined in the same order in the table declaration here as they are in the table in the database. No two columns in any given table may have the same name.

### 6.6 Variable Definition Package

The variable definition package is where you define the variables which you will be using in DML statements to hold column information for/from the database. The data types of all variables declared here must be defined in a data type definition package.

UNCLASSIFIED

A cursor is a type of variable which will be used in certain DML statements. A cursor can be defined as a pointer used in DML statements when one query may return more than one record of information. For example a select which returns more than one record of database information would have to have an associated cursor. A cursor may be reused by different queries, however, you need a separate cursor for each query returning multiple results at the same time.

Table correlation names can also be defined in the variable definition package. A correlation name is a new name assigned to a table when you want to reference the table as though it were two different tables. In the interactive queries the correlation assignment was done directly in the query. In order to use a correlation name in the DML it must be declared in either the variable definition package or in the DML package.

The format of a variable definition package is:

```
<context clause>

package <package name> is

  <use clause>

    <cursor declaration>
    ...
    <variable declaration>
    ...
    <correlation declaration>
    ...

end <package name> ;
```

Where:

<context clause> may contain "with" and "use" statements to the predefined package "DATABASE" and must contain "with" and "use" statements to the predefined package CURSOR\_DEFINITION if <cursor declaration>s are used, well as any Ada/SQL data type definition packages having referenced types in this package.

<package name> is any valid Ada package name and also must be the name of the source file.

<use clause> may "use" packages not permitted to appear in the "use" clause of the context clause such as the nested ADA\_SQL package of the data type definition package.

<cursor declaration> declares variables of cursor type for use with certain DML statements. If cursor declarations are made then the predefined package CURSOR\_DEFINITION must be "with"ed and "use"d. The format of a cursor declaration is:

```
<cursor_variable> : cursor_name;
```

where:

## UNCLASSIFIED

<cursor\_variable> is any valid Ada variable name.

<variable declaration> declares all the variables which will be used with DML statements. These include variables to hold database column data, index information for strings etc. Variable declarations are standard Ada.

<correlation declaration> declares a correlation name for a table in the format of:

```
package <new_name> is new
    <table_name>_correlation.name ( ''<new_name'' );

where:
    <new_name> is the correlation name to be assigned to the table.
    <table_name> is the database table name.
```

### 6.7 Package Body With DML Statements

Any package body may contain Ada/SQL DML statements providing certain criteria is met. We will study the various types of DML statements in later sections. The format for a package body with DML statements is:

```
<ddl context clause>
<generated package context clause>
<other context clause>
```

```
<Ada text>
...

    <cursor declaration>
    ...
    <correlation declaration>
    ...
    <Ada/SQL DML statements>
    ...
```

```
<Ada text>
...
```

Where:

<ddl context clause> may contain "with" and "use" statements to the predefined package "DATABASE" and must contain "with" and "use" statements to the predefined package CURSOR\_DEFINITION if <cursor declaration>s are used, as well as all Ada/SQL DDL definition packages referenced in this package.

UNCLASSIFIED

<generated package context clause> must contain "with" and "use" statements to the package generated by the Application Scanner. The name of this package will be that of the package or procedure of this unit with the extension of "\_ADA\_SQL".

<other context clause> may contain any other "with" and "use" clauses desired.

<Ada text> any valid Ada statements, including those required to make up a valid packages and/or procedures.

<cursor declaration> declares variables of cursor type for use with certain DML statements. If cursor declarations are made then the predefined package `CURSOR_DEFINITION` must be "with"ed and "use"d. The format of a cursor declaration is:

```
<cursor_variable> : cursor_name;
```

where:

<cursor\_variable> is any valid Ada variable name.

<correlation declaration> declares a correlation name for a table in the format of:

```
package <new_name> is new
```

```
<table_name>_correlation.name ( ''<new_name'' );
```

where:

<new\_name> is the correlation name to be assigned to the table.

<table\_name> is the database table name.

<Ada/SQL DML statements> the DML statements necessary to perform the desired interactions with the database.

## 7. The Ada Code For The DDL Units

We will now write the authorization package, the type definition package, the table definition package and the variable package which we need for our Ada/SQL program.

### 7.1 The Authorization Package

Let's write our authorization package following the format we discussed in the last section. Don't forget to with and use the predefined package "SCHEMA\_DEFINITION". Let's use "UNITED\_UNIV\_AUTH" as our authorization identifier. Name the source file "AUTH\_PACK.ADA". Your authorization package should look like this:

```
with SCHEMA_DEFINITION;
use SCHEMA_DEFINITION;

package AUTH_PACK is
  function UNITED_UNIV_AUTH is new AUTHORIZATION_IDENTIFIER;
end AUTH_PACK;
```

Now let's compile it. Have you set up a sublibrary of the Ada/SQL library to compile into? If not you must do so at this time. Now compile AUTH\_PACK.ADA. You should have no errors. If you do get an error, stop and figure out what's wrong before proceeding.

### 7.2 The Data Type Definition Package

We're now ready to create our data type definition package. We will need a data type for each and every column in our tables and for any index pointers we might have for character strings. Let's call the source file TYPES.ADA. The beginning of the package should look like:

```
package TYPES is

  package ADA_SQL is
```

Now let's add types and sub types for all the columns in our tables. We'll do it one table at a time. The DEPARTMENT table has two columns, DEPT\_ID which is a one digit integer and DEPT\_DESC which is an eight character string. Let's define type ID\_DEPARTMENT as an integer with a range of 1 to 9:

```
  type ID_DEPARTMENT is range 1 .. 9;
```

We name the type ID\_DEPARTMENT not DEPT\_ID since DEPT\_ID is the name of the column and will be the name of the record element when describing the DEPARTMENT table. For the DEPT\_DESC column let's define a constrained array of 1 to 8 CHARACTERS:

```
  type DESCRIPTION_DEPARTMENT is array (1 .. 8) of CHARACTER;
```

## UNCLASSIFIED

This is the simplest method of defining a string. You may also define the components, provided they are a subtype or a derived type of CHARACTER. You may also define the index type, providing it is of type integer. The definition of arrays may be constrained or unconstrained.

That takes care of type definitions for the DEPARTMENT table. Now let's look at the PROFESSOR table. PROF\_ID is a two digit integer:

```
type ID_PROFESSOR is range 1 .. 99;
```

Let's create a subtype of ID\_PROFESSOR with the \_NOT\_NULL\_UNIQUE suffix. This means that anything of that type must contain a value, be not null, and must be unique. We can't have a professor without an id number and they certainly must be unique.

```
subtype ID_PROFESSOR_NOT_NULL_UNIQUE is ID_PROFESSOR;
```

PROF\_NAME is a twelve character string. For this one let's define a component type of NAME\_COMPONENT and an index type of LAST\_NAME\_INDEX. And let's define a type of LAST\_NAME as the string. It should look like:

```
type NAME_COMPONENT is new CHARACTER;  
type LAST_NAME_INDEX is range 1 .. 12;  
type LAST_NAME is array (LAST_NAME_INDEX) of NAME_COMPONENT;
```

PROF\_FIRST is a ten character string. For the component type we'll use the same type defined above as NAME\_COMPONENT. For the index type we'll use FIRST\_NAME\_INDEX and we'll define the string as FIRST\_NAME. It looks like this:

```
type FIRST_NAME_INDEX is range 1 .. 10;  
type FIRST_NAME is array (FIRST_NAME_INDEX) of NAME_COMPONENT;
```

PROF\_DEPT will be the type which we defined as ID\_DEPARTMENT above. PROF\_YEARS will be an integer type with two digits which we'll call YEARS\_EMPLOYED:

```
type YEARS_EMPLOYED is range 1 .. 99;
```

PROF\_SALARY will be a floating point type named YEARLY\_INCOME with five digits before the decimal and two after it:

```
type YEARLY_INCOME is digits 7 range 0.0 .. 99999.99;
```

That takes care of the PROFESSOR table. Now let's do the COURSE table. COURSE\_ID is a three digit integer type:

```
type ID_COURSE is range 1 .. 999;
```

Let's create a subtype of ID\_COURSE which requires that data be present in the column:

```
subtype ID_COURSE_NOT_NULL is ID_COURSE;
```

COURSE\_DEPT is of the type ID\_DEPARTMENT defined above. COURSE\_DESC is a twenty character string. Let's define this type as an unconstrained array:

UNCLASSIFIED

```
type DESCRIPTION_COURSE is array (INTEGER range <>) of CHARACTER;
```

COURSE\_PROF will be of the type defined above as ID\_PROFESSOR. COURSE\_HOURS is defined as a one digit integer, but here let's make it an enumeration type. We'll define an enumeration type called ENUMERATION\_NUMBERS from zero to ten and then define SEMESTER\_HOURS as a subtype of that with a range of ONE to FIVE:

```
type ENUMERATION_NUMBERS is (ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX,  
    SEVEN, EIGHT, NINE, TEN);  
subtype SEMESTER_HOURS is ENUMERATION_NUMBERS range ONE .. FIVE ;
```

And that takes care of the PROFESSOR table. Now we'll do the STUDENT table. ST\_ID is a three digit integer :

```
type ID_STUDENT is range 1 .. 999;
```

ST\_NAME and ST\_FIRST will be of the same types which we defined for professor name above, LAST\_NAME and FIRST\_NAME. ST\_ROOM is a four character string, let's set up a general integer index type with a range of 1 through 10 and a general component type of CHARACTER and then define a general array type with is unconstrained.

```
type GENERAL_INDEX is range 1 .. 10;  
type GENERAL_COMPONENT is new CHARACTER;  
type GENERAL_ARRAY is array (GENERAL_INDEX range <>) of GENERAL_COMPONENT;
```

Let's define ST\_STATE as a subtype of GENERAL\_ARRAY with an index range of two:

```
subtype HOME_STATE is GENERAL_ARRAY (1..2);
```

ST\_MAJOR is of the type ID\_DEPARTMENT as described above. ST\_YEAR is of the type ENUMERATION\_NUMBERS with a range of ONE to FOUR:

```
subtype YEARS_ATTENDED is ENUMERATION_NUMBERS range ONE .. FOUR;
```

That finishes the STUDENT table. Now we do the CLASS table. CLASS\_STUDENT, CLASS\_DEPT and CLASS\_COURSE are defined above as ID\_STUDENT, ID\_DEPARTMENT and ID\_COURSE. We need to define a floating point type for grades to hold numbers from 0.0 through 100.0:

```
type GRADE_POINT is digits 5 range 0.0 .. 100.0;
```

CLASS\_SEM\_1, CLASS\_SEM\_2 and CLASS\_GRADE will all be of this type. And that finished the CLASS table. Now we do the GRADE TABLE. GRADE\_COURSE is of the type ID\_COURSE described above and GRADE\_AVERAGE is of the type GRADE\_POINT described above. So we don't need any new types for the GRADE table. Now for the SALARY table. SAL\_YEAR and SAL\_END are of the YEARS\_EMPLOYED type, SAL\_NIM and SAL\_MAX are of the YEARLY\_INCOME type. We have to define a type for SAL\_RAISE as a floating point number between 0.001 and 0.500.

```
type SALARY_RAISE is digits 4 range 0.001 .. 0.500 ;
```

## UNCLASSIFIED

These type definitions will be used to describe the columns in our tables as well as the variables to hold database information. The only data type that we are missing is one to hold a sum of professors salaries. When we total up several salaries the number of digits in the YEARLY\_INCOME type will not be sufficient. We want a similar field with more digits. Let's call it TOTAL\_INCOME and define it as:

```
type TOTAL_INCOME is digits 9 range 0.00 .. 99999999.00 ;
```

That's it for the type definitions. Now as a closing to the package we add:

```
end ADA_SQL;
```

```
end TYPES;
```

And that should complete our source file TYPES.ADA. Let's take a look at it all together now:

```
package TYPES is
```

```
package ADA_SQL is
```

```
type ID_DEPARTMENT is range 1 .. 9;
type DESCRIPTION_DEPARTMENT is array (1 .. 8) of CHARACTER;
type ID_PROFESSOR is range 1 .. 99;
subtype ID_PROFESSOR_NOT_NULL_UNIQUE is ID_PROFESSOR;
type NAME_COMPONENT is new CHARACTER;
type LAST_NAME_INDEX is range 1 .. 12;
type LAST_NAME is array (LAST_NAME_INDEX) of NAME_COMPONENT;
type FIRST_NAME_INDEX is range 1 .. 10;
type FIRST_NAME is array (FIRST_NAME_INDEX) of NAME_COMPONENT;
type YEARS_EMPLOYED is range 1 .. 99;
type YEARLY_INCOME is digits 7 range 0.0 .. 99999.99;
type ID_COURSE is range 1 .. 999;
subtype ID_COURSE_NOT_NULL is ID_COURSE;
type DESCRIPTION_COURSE is array (INTEGER range <>) of CHARACTER;
type ENUMERATION_NUMBERS is (ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX,
    SEVEN, EIGHT, NINE, TEN);
subtype SEMESTER_HOURS is ENUMERATION_NUMBERS range ONE .. FIVE ;
type ID_STUDENT is range 1 .. 999;
type GENERAL_INDEX is range 1 .. 10;
type GENERAL_COMPONENT is new CHARACTER;
type GENERAL_ARRAY is array (GENERAL_INDEX range <>) of GENERAL_COMPONENT;
subtype HOME_STATE is GENERAL_ARRAY (1..2);
subtype YEARS_ATTENDED is ENUMERATION_NUMBERS range ONE .. FOUR;
type GRADE_POINT is digits 5 range 0.0 .. 100.0;
type SALARY_RAISE is digits 4 range 0.001 .. 0.500 ;
type TOTAL_INCOME is digits 9 range 0.00 .. 99999999.00 ;
```

```
end ADA_SQL;
```

```
end TYPES;
```

Now compile this package. You should have no errors, however if you made a mistake correct it now



and recompile.

### 7.3 The Table Definition Package

Now we will write the source module for the table definitions. The context clause must "with" and "use" the predefined package `SCHEMA_DEFINITION` and our authorization package our type definition package. We will also want to "use" the inner package `ADA_SQL` of our type definition package. We must also declare our schema authorization since this unit will define database tables. Let's call this package `TABLES`, and the source `TABLES.ADA`. The beginning should look like this:

```
with SCHEMA_DEFINITION, AUTH_PACK, TYPES;
use SCHEMA_DEFINITION, AUTH_PACK, TYPES;

package TABLES is

    use TYPES.ADA_SQL;

    package ADA_SQL is

        SCHEMA_AUTHORIZATION : IDENTIFIER := UNITED_UNIV_AUTH;
```

We will now define the tables and their columns table by table. remember table names and column names must match exactly the names as they appear in the database. And the column types must be types defined in the `TYPES` package. We'll do one table at a time starting with the `DEPARTMENT` table. Create a record type called `DEPARTMENT` with elements `DEPT_ID` of type `ID_DEPARTMENT` and `DEPT_DESC` of type `DESCRIPTION_DEPARTMENT`:

```
type DEPARTMENT is
    record
        DEPT_ID           : ID_DEPARTMENT;
        DEPT_DESC         : DESCRIPTION_DEPARTMENT;
    end record;
```

And now the `PROFESSOR` table with column `PROF_ID` of type `ID_PROFESSOR_NOT_NULL_UNIQUE`, `PROF_NAME` of type `LAST_NAME`, `PROF_FIRST` of type `FIRST_NAME`, `PROF_DEPT` of type `ID_DEPARTMENT`, `PROF_YEARS` of type `YEARS_EMPLOYED` and `PROF_SALARY` of type `YEARLY_INCOME`.

```
type PROFESSOR is
    record
        PROF_ID           : ID_PROFESSOR_NOT_NULL_UNIQUE;
        PROF_NAME         : LAST_NAME;
        PROF_FIRST        : FIRST_NAME;
        PROF_DEPT         : ID_DEPARTMENT;
        PROF_YEARS        : YEARS_EMPLOYED;
        PROF_SALARY       : YEARLY_INCOME;
    end record;
```

UNCLASSIFIED

And now the COURSE table with column COURSE\_ID of type ID\_COURSE\_NOT\_NULL, COURSE\_DEPT of type ID\_DEPARTMENT, COURSE\_DESC of type DESCRIPTION\_COURSE (remember to add an index constraint here since the type is an unconstrained array), COURSE\_PROF of type ID\_PROFESSOR and COURSE\_HOURS of type SEMESTER\_HOURS.

```
type COURSE is
  record
    COURSE_ID      : ID_COURSE_NOT_NULL;
    COURSE_DEPT    : ID_DEPARTMENT;
    COURSE_DESC    : DESCRIPTION_COURSE (1..20);
    COURSE_PROF    : ID_PROFESSOR;
    COURSE_HOURS   : SEMESTER_HOURS;
  end record;
```

And the STUDENT table with column ST\_ID of type ID\_STUDENT, ST\_NAME of type LAST\_NAME, ST\_FIRST of type FIRST\_NAME, ST\_ROOM of type GENERAL\_ARRAY (remember to constrain the array at this time), ST\_STATE of type HOME\_STATE, ST\_MAJOR of type ID\_DEPARTMENT and ST\_YEAR of type YEARS\_ATTENDED.

```
type STUDENT is
  record
    ST_ID          : ID_STUDENT;
    ST_NAME        : LAST_NAME;
    ST_FIRST       : FIRST_NAME;
    ST_ROOM        : GENERAL_ARRAY (1..4);
    ST_STATE       : HOME_STATE;
    ST_MAJOR       : ID_DEPARTMENT;
    ST_YEAR        : YEARS_ATTENDED;
  end record;
```

The CLASS table is made of of column CLASS\_STUDENT of type ID\_STUDENT, CLASS\_DEPT of type ID\_DEPARTMENT, CLASS\_COURSE of type ID\_COURSE, CLASS\_SEM\_1 of type GRADE\_POINT, CLASS\_SEM\_2 of type GRADE\_POINT and CLASS\_GRADE of type GRADE\_POINT.

```
type CLASS is
  record
    CLASS_STUDENT  : ID_STUDENT;
    CLASS_DEPT     : ID_DEPARTMENT;
    CLASS_COURSE   : ID_COURSE;
    CLASS_SEM_1    : GRADE_POINT;
    CLASS_SEM_2    : GRADE_POINT;
    CLASS_GRADE    : GRADE_POINT;
  end record;
```

The GRADE table is made up of column GRADE\_COURSE of type ID\_COURSE and GRADE\_AVERAGE of type GRADE\_POINT.

```
type GRADE is
  record
```

UNCLASSIFIED

```
    GRADE_COURSE      : ID_COURSE;  
    GRADE_AVERAGE    : GRADE_POINT;  
end record;
```

The SALARY table is made up of column SAL\_YEAR of type YEARS\_EMPLOYED, SAL\_END of type YEARS\_EMPLOYED, SAL\_MIN of type YEARLY\_INCOME, SAL\_MAX of type YEARLY\_INCOME and SAL\_RAISE of type SALARY\_RAISE.

```
type SALARY is  
  record  
    SAL_YEAR          : YEARS_EMPLOYED;  
    SAL_END            : YEARS_EMPLOYED;  
    SAL_MIN            : YEARLY_INCOME;  
    SAL_MAX            : YEARLY_INCOME;  
    SAL_RAISE          : SALARY_RAISE;  
  end record;
```

And we end the module with:

```
end ADA_SQL;
```

```
end TABLES;
```

Put it all together and we get:

```
with SCHEMA_DEFINITION, AUTH_PACK, TYPES;  
use SCHEMA_DEFINITION, AUTH_PACK, TYPES;
```

```
package TABLES is
```

```
  use TYPES.ADA_SQL;
```

```
  package ADA_SQL is
```

```
    SCHEMA_AUTHORIZATION : IDENTIFIER := UNITED_UNIV_AUTH;
```

```
  type DEPARTMENT is
```

```
    record  
      DEPT_ID          : ID_DEPARTMENT;  
      DEPT_DESC        : DESCRIPTION_DEPARTMENT;  
    end record;
```

```
  type PROFESSOR is
```

```
    record  
      PROF_ID          : ID_PROFESSOR_NOT_NULL_UNIQUE;  
      PROF_NAME        : LAST_NAME;  
      PROF_FIRST       : FIRST_NAME;  
      PROF_DEPT        : ID_DEPARTMENT;  
      PROF_YEARS       : YEARS_EMPLOYED;  
      PROF_SALARY      : YEARLY_INCOME;  
    end record;
```

# UNCLASSIFIED

```

type COURSE is
  record
    COURSE_ID      : ID_COURSE_NOT_NULL;
    COURSE_DEPT    : ID_DEPARTMENT;
    COURSE_DESC    : DESCRIPTION_COURSE (1..20);
    COURSE_PROF    : ID_PROFESSOR;
    COURSE_HOURS   : SEMESTER_HOURS;
  end record;

```

```

type STUDENT is
  record
    ST_ID          : ID_STUDENT;
    ST_NAME        : LAST_NAME;
    ST_FIRST       : FIRST_NAME;
    ST_ROOM        : GENERAL_ARRAY (1..4);
    ST_STATE       : HOME_STATE;
    ST_MAJOR       : ID_DEPARTMENT;
    ST_YEAR        : YEARS_ATTENDED;
  end record;

```

```

type CLASS is
  record
    CLASS_STUDENT  : ID_STUDENT;
    CLASS_DEPT     : ID_DEPARTMENT;
    CLASS_COURSE   : ID_COURSE;
    CLASS_SEM_1    : GRADE_POINT;
    CLASS_SEM_2    : GRADE_POINT;
    CLASS_GRADE    : GRADE_POINT;
  end record;

```

```

type GRADE is
  record
    GRADE_COURSE   : ID_COURSE;
    GRADE_AVERAGE : GRADE_POINT;
  end record;

```

```

type SALARY is
  record
    SAL_YEAR       : YEARS_EMPLOYED;
    SAL_END        : YEARS_EMPLOYED;
    SAL_MIN        : YEARLY_INCOME;
    SAL_MAX        : YEARLY_INCOME;
    SAL_RAISE      : SALARY_RAISE;
  end record;

```

end ADA\_SQL;

end TABLES;

Now compile this package. You should have no errors, however if you made a mistake correct it now and recompile.

## 7.4 The Variable Definition Package

It's now time to write our variable definition package. We want variables for every column of data we will manipulate in the database. We'll also need variables to hold index pointers for the strings. We may need additional variables at a later date but for now let's define one variable per column and one index pointer per string. We'll name our variables the same as the column names but with a V\_ prefix. We'll name the index pointers the same as the column names but with a V\_ prefix and a \_INDEX suffix. The variables are not included in an inner package named ADA\_SQL and no authorization identifier is needed. We will need a cursor variable also, we'll call it CURSOR, so remember to "with" and "use" the predefined package CURSOR\_DEFINITION. You should end up with a package called VARIABLES, remember to name the source file VARIABLES.ADA, which looks like:

```
with TYPES, CURSOR_DEFINITION, DATABASE;
use CURSOR_DEFINITION;

package VARIABLES is

  use TYPES.ADA_SQL;

  CURSOR                : CURSOR_NAME;

  V_DEPT_ID              : ID_DEPARTMENT;
  V_DEPT_DESC            : DESCRIPTION_DEPARTMENT;
  V_DEPT_DESC_INDEX      : INTEGER;

  V_PROF_ID              : ID_PROFESSOR;
  V_PROF_NAME            : LAST_NAME;
  V_PROF_NAME_INDEX      : LAST_NAME_INDEX;
  V_PROF_FIRST           : FIRST_NAME;
  V_PROF_FIRST_INDEX     : FIRST_NAME_INDEX;
  V_PROF_DEPT            : ID_DEPARTMENT;
  V_PROF_YEARS           : YEARS_EMPLOYED;
  V_PROF_SALARY          : YEARLY_INCOME;

  V_COURSE_ID            : ID_COURSE;
  V_COURSE_DEPT          : ID_DEPARTMENT;
  V_COURSE_DESC          : DESCRIPTION_COURSE (1..20);
  V_COURSE_DESC_INDEX    : INTEGER;
  V_COURSE_PROF          : ID_PROFESSOR;
  V_COURSE_HOURS         : SEMESTER_HOURS;

  V_ST_ID                : ID_STUDENT;
  V_ST_NAME              : LAST_NAME;
  V_ST_NAME_INDEX        : LAST_NAME_INDEX;
  V_ST_FIRST             : FIRST_NAME;
  V_ST_FIRST_INDEX       : FIRST_NAME_INDEX;
  V_ST_ROOM              : GENERAL_ARRAY (1..4);
  V_ST_ROOM_INDEX        : GENERAL_INDEX;
  V_ST_STATE             : HOME_STATE;
  V_ST_STATE_INDEX       : GENERAL_INDEX;
```

UNCLASSIFIED

V\_ST\_MAJOR : ID\_DEPARTMENT;  
V\_ST\_YEAR : YEARS\_ATTENDED;

V\_CLASS\_STUDENT : ID\_STUDENT;  
V\_CLASS\_DEPT : ID\_DEPARTMENT;  
V\_CLASS\_COURSE : ID\_COURSE;  
V\_CLASS\_SEM\_1 : GRADE\_POINT;  
V\_CLASS\_SEM\_2 : GRADE\_POINT;  
V\_CLASS\_GRADE : GRADE\_POINT;

V\_GRADE\_COURSE : ID\_COURSE;  
V\_GRADE\_AVERAGE : GRADE\_POINT;

V\_SAL\_YEAR : YEARS\_EMPLOYED;  
V\_SAL\_END : YEARS\_EMPLOYED;  
V\_SAL\_MIN : YEARLY\_INCOME;  
V\_SAL\_MAX : YEARLY\_INCOME;  
V\_SAL\_RAISE : SALARY\_RAISE;

COUNT\_RESULT : DATABASE.INT;  
AVG\_SALARY : YEARLY\_INCOME;  
MIN\_SALARY : YEARLY\_INCOME;  
MAX\_SALARY : YEARLY\_INCOME;  
SUM\_SALARY : TOTAL\_INCOME;  
AVG\_SEM\_1 : GRADE\_POINT;  
AVG\_SEM\_2 : GRADE\_POINT;

end VARIABLES;

Now compile it and fix any errors you may have.

## 8. The Basics For The Ada/SQL Program

In this section we will write the DML source code for our Ada/SQL program. First we will write a skeleton DML module. Example by example we'll build the full DML unit. When I show an example I will only show the section of the DML unit pertaining to that particular example. Each time we run an example, run the DML unit with only that one example in it. But be sure to save all the examples together in another file and by the end of this section you'll have an Ada/SQL application program which runs many many queries.

### 8.1 Conversion Subroutines

As we run each example we will want to print out the results of the queries. We would like to use the PUT routines from the standard Ada package TEXT\_IO. To do this we must instantiate numerous generic functions with the data types defined in TYPES.ADA. We also must write our own generic function to handle our character strings. The packages I'll be using to do all of my data type conversions for the purpose of printing the information on the terminal is shown below. The source file is called CONVERSIONS.ADA. You may use these packages or generate something of your own. This package is standard Ada so I'm not going to discuss it any further.

```
with TYPES, TEXT_IO;
use TEXT_IO;

package CONVERSION_SUBS is

  use TYPES.ADA_SQL;

  -- each different type of component for arrays needs a routine to convert
  -- the individual components to CHARACTER components of a STRING

  function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in NAME_COMPONENT)
    return CHARACTER;

  function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in GENERAL_COMPONENT)
    return CHARACTER;

  function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in CHARACTER)
    return CHARACTER;

  -- we need a generic routine for the conversion of all constrained arrays
  -- to STRINGS

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type ARRAY_TYPE is array ( INDEX_TYPE ) of COMPONENT_TYPE;
    with function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
```

# UNCLASSIFIED

```

        ( CHAR_IN : COMPONENT_TYPE )
        return CHARACTER is <>;
package CONSTRAINED_ARRAYS_IO is

-- we call the routine PUT so in the DML program it's transparent that all
-- kinds of conversions are taking place

procedure PUT
    (STRING_IN  : in ARRAY_TYPE;
     INDEX_IN   : in INDEX_TYPE);

end CONSTRAINED_ARRAYS_IO;

-- we need a generic routine for the conversion of all unconstrained arrays
-- to STRINGS

generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type ARRAY_TYPE is array ( INDEX_TYPE range <> ) of COMPONENT_TYPE;
    with function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
        ( CHAR_IN : COMPONENT_TYPE )
        return CHARACTER is <>;
package UNCONSTRAINED_ARRAYS_IO is

-- we call the routine PUT so in the DML program it's transparent that all
-- kinds of conversions are taking place

procedure PUT
    (STRING_IN  : in ARRAY_TYPE;
     INDEX_IN   : in INDEX_TYPE);
end UNCONSTRAINED_ARRAYS_IO;

end CONVERSION_SUBS;

package body CONVERSION_SUBS is

-- each different type of component for arrays needs a routine to convert
-- the individual components to CHARACTER components of a STRING

function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in NAME_COMPONENT)
    return CHARACTER is
begin
    return CHARACTER (CHAR_IN);
end CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS;

function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in GENERAL_COMPONENT)
    return CHARACTER is
begin

```



UNCLASSIFIED

```
        return CHARACTER (CHAR_IN);
    end CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS;

    function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
        (CHAR_IN : in CHARACTER)
        return CHARACTER is
    begin
        return CHARACTER (CHAR_IN);
    end CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS;

-- we need a generic routine for the conversion of all constrained arrays
-- to STRINGS

package body CONSTRAINED_ARRAYS_IO is

-- we call the routine PUT so in the DML program it's transparent that all
-- kinds of conversions are taking place

    procedure PUT
        (STRING_IN  : in ARRAY_TYPE;
         INDEX_IN    : in INDEX_TYPE) is

        STRING_OUT : STRING (1..100);
        INDEX_OUT  : INTEGER;

    begin
        INDEX_OUT := INTEGER (INDEX_IN);
        for I in 1.. INTEGER (INDEX_IN)
        loop
            STRING_OUT (I) := CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
                               (STRING_IN (INDEX_TYPE (I))) ;
        end loop;
        PUT (STRING_OUT (1..INDEX_OUT));
    end PUT;

end CONSTRAINED_ARRAYS_IO;

-- we need a generic routine for the conversion of all unconstrained arrays
-- to STRINGS

package body UNCONSTRAINED_ARRAYS_IO is

-- we call the routine PUT so in the DML program it's transparent that all
-- kinds of conversions are taking place

    procedure PUT
        (STRING_IN  : in ARRAY_TYPE;
         INDEX_IN    : in INDEX_TYPE) is

        STRING_OUT : STRING (1..100);
        INDEX_OUT  : INTEGER;
```

# UNCLASSIFIED

```

begin
  INDEX_OUT := INTEGER (INDEX_IN);
  for I in 1.. INTEGER (INDEX_IN)
  loop
    STRING_OUT (I) := CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
      (STRING_IN (INDEX_TYPE (I))) ;
  end loop;
  PUT (STRING_OUT (1..INDEX_OUT));
end PUT;

end UNCONSTRAINED_ARRAYS_IO;

end CONVERSION_SUBS;

-- the package CONVERSIONS instantiates all the necessary packages so our
-- DML unit can "with" and "use" them and not be cluttered with all this
-- instantiation

with TEXT_IO, TYPES, CONVERSION_SUBS, DATABASE;
use TEXT_IO, CONVERSION_SUBS;

package CONVERSIONS is

  use TYPES.ADA_SQL;

  -- to output character strings

  package CONVERT_LAST_NAME is new CONSTRAINED_ARRAYS_IO
    (LAST_NAME_INDEX, NAME_COMPONENT, LAST_NAME);

  package CONVERT_FIRST_NAME is new CONSTRAINED_ARRAYS_IO
    (FIRST_NAME_INDEX, NAME_COMPONENT, FIRST_NAME);

  package CONVERT_DESCRIPTION_COURSE is new UNCONSTRAINED_ARRAYS_IO
    (INTEGER, CHARACTER, DESCRIPTION_COURSE);

  package CONVERT_GENERAL_ARRAY is new UNCONSTRAINED_ARRAYS_IO
    (GENERAL_INDEX, GENERAL_COMPONENT, GENERAL_ARRAY);

  -- to output integer data as strings

  package I1_CONVERT is new INTEGER_IO (ID_DEPARTMENT);
  package I2_CONVERT is new INTEGER_IO (ID_PROFESSOR);
  package I3_CONVERT is new INTEGER_IO (YEARS_EMPLOYED);
  package I4_CONVERT is new INTEGER_IO (ID_COURSE);
  package I5_CONVERT is new INTEGER_IO (ID_STUDENT);
  package I6_CONVERT is new INTEGER_IO (DATABASE.INT);

  -- to output floating point data as strings

```

## UNCLASSIFIED

```
package CONVERT_FLOAT_YEARLY_INCOME is new FLOAT_IO (YEARLY_INCOME);
package CONVERT_FLOAT_GRADE_POINT is new FLOAT_IO (GRADE_POINT);
package CONVERT_FLOAT_SALARY_RAISE is new FLOAT_IO (SALARY_RAISE);
package CONVERT_FLOAT_TOTAL_INCOME is new FLOAT_IO (TOTAL_INCOME);

-- to output enumeration data as strings

package CONVERT_ENUMERATION_ENUMERATION_NUMBERS is new
    ENUMERATION_IO (ENUMERATION_NUMBERS);

-- to output stuff from text_io since we can't "use" text_io cause COUNT is
-- redundant as in select count (*)

procedure PUT_LINE (ITEM : in STRING ) renames TEXT_IO.PUT_LINE;
procedure NEW_LINE (SPACING : in TEXT_IO.POSITIVE_COUNT := 1)
    renames TEXT_IO.NEW_LINE;
procedure SET_COL (TO : in TEXT_IO.POSITIVE_COUNT) renames TEXT_IO.SET_COL;
procedure PUT (ITEM : in STRING ) renames TEXT_IO.PUT;

end CONVERSIONS;
```

Compile this unit or the unit you intend to use for conversions at this time.

### 8.2 Exceptions Likely To Be Encountered

There are three exceptions which you will be likely to encounter. They are `NO_UPDATE_ERROR`, which means the modification you attempted to make to database was not performed, `NOT_FOUND_ERROR`, which means that no records were selected from your query or that all records selected have been returned to you and `UNIQUE_ERROR`, which means that a query which should have selected only one record selected more than one. There are other self-explanatory exceptions which indicate serious problems, such as `INTERNAL_ERROR` and `UNDEFINED_RDBMS_ERROR`, which means the DBMS detected an error and could not execute the query. The three mentioned above are the ones you'll want to trap after queries.

### 8.3 Skeleton Of The DML Unit

We will now write a skeleton of the DML source program. Let's call our package `EXAMPLES` and our source file `EXAMPLES.ADA`. The first context clause must contain references to all the DDL units needed. This will be `TYPES`, `TABLES` and `VARIABLES`. The next context clause must be a reference to the package which will be generated by the Application Scanner. The name of this package will be `EXAMPLES_ADA_SQL`. We will include `TEXT_IO`, Ada's standard IO package and our package `CONVERSIONS` in the next context clause. `TEXT_IO` should be "with"ed but it must not be "use"ed in an Ada/SQL program. It has definitions which conflict with reserved works in Ada/SQL, namely `COUNT`. We then name a procedure and "use" the `ADA_SQL` inner package of `TYPES`. This is where you would declare correlation associations. We will not include them here until we reach an example which uses them. You will also have to "use" the instantiated functions from the `CONVERSIONS` package. You can declare `CURSOR` variables or other variables needed here, except those

UNCLASSIFIED

associated with table columns. For now we will include only one variable here, GOT\_ONE a natural which we will use to count the number of records returned by queries. Any Ada code you wish may be included in the procedure, including DML statements.

For now, before we learn any DML statements, let's write some quick code in the EXAMPLES procedure. We'll fill the variables associated with the columns of the various tables and then print them out on the terminal. We will be using this code to print all of our records, with column headings, throughout the examples. This gives you a chance to see how the conversions work for printing, or to test your conversions if you chose not to use mine. Include exception trapping for the three common exceptions. The following is the DML source we will begin with:

```
-- first context clause MUST contain the DDL packages

with TYPES, TABLES, VARIABLES;
  use TYPES, TABLES, VARIABLES;

-- second context clause MUST contain the generated package, who's name will
-- be the same as the DML package with the extension _ADA_SQL

with EXAMPLES_ADA_SQL;
  use EXAMPLES_ADA_SQL;

-- subsequent context clauses may contain any other packages you desire
-- remember not to "use" TEXT_IO due to it's conflict with reserved words

with TEXT_IO, CONVERSIONS;
  use CONVERSIONS;

procedure EXAMPLES is

  use TYPES.ADA_SQL;

-- to do all the data conversions for displaying information

  use CONVERT_LAST_NAME, CONVERT_FIRST_NAME, CONVERT_DESCRIPTION_COURSE,
    CONVERT_GENERAL_ARRAY, I1_CONVERT, I2_CONVERT, I3_CONVERT, I4_CONVERT,
    I5_CONVERT, I6_CONVERT, CONVERT_FLOAT_YEARLY_INCOME,
    CONVERT_FLOAT_TOTAL_INCOME, CONVERT_FLOAT_GRADE_POINT,
    CONVERT_FLOAT_SALARY_RAISE, CONVERT_ENUMERATION_ENUMERATION_NUMBERS;

  GOT_ONE : NATURAL := 0;

begin

-- fill the variables associated with the DEPARTMENT table
-- fill character strings to their maximum width

  V_DEPT_ID           := 1;
  V_DEPT_DESC         := "History ";
  V_DEPT_DESC_INDEX   := 8;
```

# UNCLASSIFIED

-- fill the variables associated with the PROFESSOR table

```
V_PROF_ID      := 1;
V_PROF_NAME    := "Dysart      ";
V_PROF_NAME_INDEX := 12;
V_PROF_FIRST   := "Gregory    ";
V_PROF_FIRST_INDEX := 10;
V_PROF_DEPT    := 3;
V_PROF_YEARS   := 03;
V_PROF_SALARY  := 35000.00;
```

-- fill the variables associated with the COURSE table

-- COURSE\_HOURS is defined as an enumeration type

```
V_COURSE_ID      := 101;
V_COURSE_DEPT    := 1;
V_COURSE_DESC    := "World History      ";
V_COURSE_DESC_INDEX := 20;
V_COURSE_PROF    := 05;
V_COURSE_HOURS   := TWO;
```

-- fill the variables associated with the STUDENT table

-- ST\_YEAR is defined as an enumeration type

```
V_ST_ID      := 1;
V_ST_NAME    := "Horrigan    ";
V_ST_NAME_INDEX := 12;
V_ST_FIRST   := "William    ";
V_ST_FIRST_INDEX := 10;
V_ST_ROOM    := "A101";
V_ST_ROOM_INDEX := 4;
V_ST_STATE   := "VA";
V_ST_STATE_INDEX := 2;
V_ST_MAJOR   := 3;
V_ST_YEAR    := FOUR;
```

-- fill the variables associated with the CLASS table

```
V_CLASS_STUDENT := 001;
V_CLASS_DEPT    := 3;
V_CLASS_COURSE  := 302;
V_CLASS_SEM_1   := 089.49;
V_CLASS_SEM_2   := 051.91;
V_CLASS_GRADE   := 000.00;
```

-- fill the variables associated with the GRADE table

```
V_GRADE_COURSE := 502;
V_GRADE_AVERAGE := 99.99;
```

-- fill the variables associated with the SALARY table

# UNCLASSIFIED

```

V_SAL_YEAR      := 1;
V_SAL_END       := 1;
V_SAL_MIN       := 20000.00;
V_SAL_MAX       := 29999.99;
V_SAL_RAISE     := 0.010;

-- to output integers the PUT routine which we created in the CONVERSIONS
-- package needs the variable and the width of the field.

-- to output floats the PUT routine which we created in the CONVERSIONS
-- package needs the variable, the number of digits before the decimal, the
-- number of digits after the decimal and a zero to indicate no exponent

-- to output strings the PUT routine which we created in the CONVERSIONS
-- package needs the variable and the length of the field.

-- NOTE: V_DEPT_DESC gets handled a bit differently since its index type
-- defaults and it's component type is CHARACTER, so it doesn't get an
-- instantiated generic function

-- to output enumerations the PUT routine which we created in the
-- CONVERSIONS package needs only the variable

-- now print out the variables of the DEPARTMENT table, be sure to include
-- headers and spacing

NEW_LINE;
PUT_LINE ("DEPT_ID  DEPT_DESC");
SET_COL (1); -- DEPT_ID
  PUT (V_DEPT_ID, 1);
SET_COL (11); -- DEPT_DESC
  PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
NEW_LINE;

-- now print out the variables of the PROFESSOR table

NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
          "PROF_YEARS  PROF_SALARY");
SET_COL (1); -- PROF_ID
  PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
  PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
  PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
  PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);

```

UNCLASSIFIED

```

NEW_LINE;

-- now print out the variables of the COURSE table

NEW_LINE;
PUT_LINE ("COURSE_ID  COURSE_DEPT  COURSE_DESC           " &
          "COURSE_PROF  COURSE_HOURS");
SET_COL (1);  -- COURSE_ID
  PUT (V_COURSE_ID, 3);
SET_COL (12); -- COURSE_DEPT
  PUT (V_COURSE_DEPT, 1);
SET_COL (24); -- COURSE_DESC
  PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
SET_COL (46); -- COURSE_PROF
  PUT (V_COURSE_PROF, 2);
SET_COL (58); -- COURSE_HOURS
  PUT (V_COURSE_HOURS);
NEW_LINE;

-- now print out the variables of the STUDENT table

NEW_LINE;
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
     "ST_MAJOR  ST_YEAR");
SET_COL (1);  -- ST_ID
  PUT (V_ST_ID, 3);
SET_COL (8);  -- ST_NAME
  PUT (V_ST_NAME, V_ST_NAME_INDEX);
SET_COL (22); -- ST_FIRST
  PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
SET_COL (34); -- ST_ROOM
  PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
SET_COL (43); -- ST_STATE
  PUT (V_ST_STATE, V_ST_STATE_INDEX);
SET_COL (53); -- ST_MAJOR
  PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
  PUT (V_ST_YEAR);
NEW_LINE;

-- now print out the variables of the CLASS table

NEW_LINE;
PUT_LINE ("CLASS_STUDENT CLASS_DEPT CLASS_COURSE CLASS_SEM_1 " &
          "CLASS_SEM_2 CLASS_GRADE");
SET_COL (1);  -- CLASS_STUDENT
  PUT (V_CLASS_STUDENT, 3);
SET_COL (15); -- CLASS_DEPT
  PUT (V_CLASS_DEPT, 1);
SET_COL (26); -- CLASS_COURSE
  PUT (V_CLASS_COURSE, 3);

```

# UNCLASSIFIED

```

SET_COL (39); -- CLASS_SEM_1
PUT (V_CLASS_SEM_1, 3, 2, 0);
SET_COL (51); -- CLASS_SEM_2
PUT (V_CLASS_SEM_2, 3, 2, 0);
SET_COL (63); -- CLASS_GRADE
PUT (V_CLASS_GRADE, 3, 2, 0);
NEW_LINE;

-- now print out the variables of the GRADE table

NEW_LINE;
PUT_LINE ("GRADE_COURSE      GRADE_AVERAGE");
SET_COL (1); -- GRADE_COURSE
PUT (V_GRADE_COURSE, 3);
SET_COL (19); -- GRADE_AVERAGE
PUT (V_GRADE_AVERAGE, 3, 2, 0);
NEW_LINE;

-- now print out the variables of the SALARY table

NEW_LINE;
PUT_LINE ("SAL_YEAR  SAL_END  SAL_MIN  SAL_MAX");
SET_COL (1); -- SAL_YEAR
PUT (V_SAL_YEAR, 2);
SET_COL (11); -- SAL_END
PUT (V_SAL_END, 2);
SET_COL (20); -- SAL_MIN
PUT (V_SAL_MIN, 5, 2, 0);
SET_COL (30); -- SAL_MAX
PUT (V_SAL_MAX, 5, 2, 0);
NEW_LINE;

exception
when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");

end EXAMPLES_1;

```

Before EXAMPLES.ADA can be compiled you will have to run the Application Scanner to create the generated package EXAMPLES\_ADA\_SQL.ADA. Go ahead and do this. It will tell you if there are any errors in your DDL modules or in EXAMPLES.ADA. If you have errors, correct them and rerun the scanner. When you have an error free scanner run, compile EXAMPLES\_ADA\_SQL.ADA, then compile EXAMPLES.ADA. Your entire program is now compiled and it is time to link it so we can execute it. Your link procedure may be somewhat different than what you're used to since you will have to include libraries etc. for your DBMS. If you do not know how to do this ask your database or system person. Now link EXAMPLES. You should be able to run the program and see the results. Go ahead and run it now:

```
DEPT_ID  DEPT_DESC
```



# UNCLASSIFIED

## 1 History

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00

COURSE_ID	COURSE_DEPT	COURSE_DESC	COURSE_PROF	COURSE_HOURS
101	1	World History	5	TWO

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR

CLASS_STUDENT	CLASS_DEPT	CLASS_COURSE	CLASS_SEM_1	CLASS_SEM_2	CLASS_GRADE
1	3	302	89.49	51.91	0.00

GRADE_COURSE	GRADE_AVERAGE
502	99.99

SAL_YEAR	SAL_END	SAL_MIN	SAL_MAX
1	1	20000.00	29999.99

Did you get the same results that I did? If not figure out what is different, make adjustments and run it again. Remember every time that you make a change to EXAMPLES.ADA you will have to run the Application Scanner and then compile both EXAMPLES\_ADA\_SQL.ADA and EXAMPLES.ADA. Make sure you're comfortable with the modules we've written so far, with the scanner, with the compilation procedure and with the link procedure. You may wish to save the convert and print routines in as a stencil to use for printing the results of the queries. In the next section we will write DML statements and insert them into EXAMPLES.ADA in place of filling the variables and printing the columns.

## 9. Getting Ready For Ada/SQL DML Queries

I will now show you the Ada/SQL DML queries corresponding to the queries we looked at in the interactive section. Queries in the interactive mode immediately return results when they are executed. Ada/SQL queries are embedded in a program and return results into variables. Your program must extract that data and format it for printing. In the examples we will be using the same basic Ada/SQL DML unit which we wrote in section 8.3 above. Remove everything after the "begin" and the "end EXAMPLES" and insert the DML to create a DML unit which can be scanned, compiled, linked and run. Most of the queries we will do one at a time. Insert only that query into the framework of the EXAMPLES program and run it. You may wish to save all of the queries to put together into one large program at the end. At this time your tables should be defined to the DBMS. Ada/SQL manipulates data within existing tables, it is not used to create tables. Your tables should be empty at this time. If you have just completed the interactive queries your tables will be empty. It would be a good idea to check at this time and delete any existing data in the tables.

### 9.1 OPEN\_DATABASE

The OPEN\_DATABASE statement identifies you to the DBMS. It must be the first Ada/SQL DML

statement executed in a program. You must supply your ID and password to the DBMS in order to be allowed to manipulate any data within the tables. Upon receiving the OPEN\_DATABASE command the DBMS will prepare the database for use by your program. Exactly what takes place in the DBMS will differ from one DBMS to another. For example, a DBMS may flag a database as being in use and not allow another user to log on to the same database to avoid a conflict with two users updating the same database at the same time. The format of the OPEN\_DATABASE statement is :

```
OPEN_DATABASE ("user_id","password");
```

You probably needed a user ID and password to log on to the DBMS when we did interactive queries. If so they will be the same ones used here. If not ask your database or system personnel for the appropriate id and password.

In our program EXAMPLES, immediately after the begin statement add a OPEN\_DATABASE statement, using your id and password. For example:

```
OPEN_DATABASE ("UNIVERSITY","SECRET_CODE");
```

## 9.2 EXIT\_DATABASE

The EXIT\_DATABASE statement signals the DBMS that you have completed your transactions and now wish to log off. The EXIT\_DATABASE statement must be the last Ada/SQL DML statement in a program. Upon receiving an EXIT\_DATABASE command the DBMS will do some clean up routines. These will be determined by the specific underlying DBMS. Many DBMS will back out all changes made to the database if an EXIT\_DATABASE statement is not executed. For example if your program aborts in the middle of execution it is likely that your program will not have inflicted any changes to the database. Again this is DBMS specific, find out how your DBMS will handle such a situation.

The format of the EXIT\_DATABASE is simply:

```
EXIT_DATABASE;
```

We need to add such a statement at the end of our EXAMPLES program. The line before "end EXAMPLES;" should now read:

```
EXIT_DATABASE;
```

## 9.3 Cursors

When using interactive queries the results were automatically formatted and printed out on the screen. When using Ada/SQL the results of a query will be stored in variables. You will fetch the information from one record at a time. When you are dealing with a query which returns multiple records you will need a pointer to the results of the query which will cause the records to be returned to you in the correct order. We use a data structure called a cursor for this purpose. In Ada/SQL all cursors must be of data type CURSOR\_DEFINITION.CURSOR\_NAME. We will be using a cursor in our DML examples which we defined in the VARIABLES package simply as CURSOR. A cursor is used only

with a select statement in which you plan to retrieve multiple records.

When using a cursor you will first declare the cursor by associating a query with it. To do this we use the DECLAR statement. Note the spelling, it is DECLAR not DECLARE since DECLARE is a reserved word and may not be used here. The format of the DECLAR statement is:

```
DECLAR ( cursor_name , CURSOR_FOR =>
        select_statement ) ;
```

Cursor\_name is the name of the cursor variable. We have used CURSOR for this purposes. Select\_statement is any statement retrieving data. We will get to select statements in a moment. A select statement within a DECLAR may return any number of records. A select statement MUST be within a DECLAR statement if it returns more than one record. The execution of a select statement not within a DECLAR statement which returns multiple records will result in the exception "UNIQUE\_ERROR".

Once the cursor has been declared and associated with a query an OPEN statement is executed. The OPEN statement will cause the query to be executed and will prepare output to be returned to you via the FETCH statement (more on FETCH in a moment). The format of the OPEN statement is simply:

```
OPEN ( cursor_name );
```

When you are finished with a query you issue a CLOSE cursor statement which cleans up and releases that cursor for future use. Once the CLOSE cursor has been executed you can no longer retrieve any data from the query associated with that cursor. A CLOSE statement may be issued before you have retrieved all records if and only if you do not wish to continue retrieving the records. The format of the CLOSE statement is:

```
CLOSE ( cursor_name );
```

You may define multiple cursors for the purpose of executing multiple data selections simultaneously. For example you may wish to receive data from two different tables and merge the output. Once you have closed a cursor it may be redeclared again with a new query. The declar, open and close statements of a query must all reference the same cursor.

## 10. Ada/SQL DML Queries

We are now ready to process the same queries which we used for examples in section 4 as Ada/SQL DML queries. You should have the framework for EXAMPLES.ADA set up. Each query should be between the OPEN\_DATABASE and EXIT\_DATABASE statement. After writing each query, run EXAMPLES.ADA through the Application Scanner, check for errors, compile the generated package then EXAMPLES.ADA, link the program and run it to see the results. The queries done here are the same ones done interactively in section 4 and results should be the same.

## 10.1 SELECT & FROM & FETCH & INTO

To retrieve information from one or more tables you will use a SELEC statement (note spelling SELEC not SELECT, since SELECT is reserved word) which specifies the columns you wish to see and a FROM clause to indicate the tables from which to extract the column data. An asterisk enclosed in single quotes ('\*') may be used in place of column names to indicate all column names in the table. The columns will be displayed in the order stated in the select clause, if all columns are selected with the asterisk then the columns will be displayed in the order stated in the create table command. The format of the select clause is:

```
SELEC ( column_1 & column_2 & ... ,
      FROM => table & table & ... ) ;
or
SELEC ( '*' ,
      FROM => table & table & ... ) ;
```

A SELEC statement which will return more than one record must be enclosed in a DECLAR cursor statement. It must also have the appropriate OPEN cursor and CLOSE cursor statements following it. The format of a SELEC statement within a DECLAR statement is:

```
DECLAR ( cursor, CURSOR_FOR =>
      SELEC ( column_1 & column_2 & ... ,
      FROM => table & table & ... ) ) ;
```

All punctuation is very important and must be used as shown.

To fill the program variables with data from a query you use the FETCH and INTO statements. The FETCH statement is used only with the DECLAR cursor statement. When not DECLARING a cursor you do not use the FETCH statement since you will only be retrieving the data from one record. The FETCH statement causes the next available record to be readied for insertion into your variables. You will issue one FETCH statement per record retrieved. A FETCH will result in a NOT\_FOUND\_ERROR exception when there are no more records to be returned. This will occur on the first FETCH if no records were selected by the query. For this reason we keep count of the number of records returned in our EXAMPLES program with the variable GOT\_ONE so that when we encounter the NOT\_FOUND\_ERROR exception we know if we've reached the end of the records or if we never selected any records. If no records are selected from a query not within a DECLAR statement a NOT\_FOUND\_ERROR exception will be issued from the OPEN cursor statement. The format of the FETCH statement is:

```
FETCH ( cursor_name ) ;
```

The FETCH statement must appear after an OPEN cursor statement and before a CLOSE cursor statement. It will most likely be in a loop so you can fetch all records and in a block so you can trap exceptions.

The INTO statements result in the moving of the data into your variables. You will use one INTO statement for each column listed in the SELEC statement. The INTO statements when used with a DECLAR cursor must immediately follow the FETCH, and when used outside of a DECLAR cursor must immediately follow the SELEC statement. Nothing must fall between the FETCH and INTO or the SELEC and INTO or you may get strange errors on your compilation about not being able to find an

UNCLASSIFIED

appropriate INTO routine. This is because unless the INTO statements immediately follow the FETCH or SELEC statement the Application Scanner will not pick them up and will not create the appropriate INTO routines. There are two formats for the INTO statement. For all column types except character strings the format is:

```
INTO ( variable_name ) ;
```

For character strings the format of the INTO statement is:

```
INTO ( variable_name, variable_index ) ;
```

After the completion of the INTO statement the variable\_name will contain the data for the column. The variable\_index will contain the last used character element in the character string variable\_name. There will be one INTO statement for each column selected in the query. These INTO statements must be in the same order in which the columns are selected. Variable\_name must be of the same data type as the column who's data it is receiving. Variable\_index must be of the same data type as the index of the character string variable\_name. You will receive a constraint error if you try to fill a variable with data deemed unacceptable to Ada. Therefore be sure character strings are padded with spaces, blank numeric fields have a zero or another number in them, and that all constraints and ranges placed by the data type are met when retrieving data.

You will place your FETCH and INTOs in a loop when retrieving multiple records. You will also place this loop inside a block for the purpose of trapping exceptions. In the example below note the formatting we use, as a comment I will show the equivalent interactive query, I will print out this example number so we can see what's happening as the program executes. For a query returning a single result I will then start a block, issue the SELEC statement, the INTO statements, print the results, check for exceptions and end the block. For a query returning multiple results I will issue a DECLAR statement the SELEC statement, then start a block and a loop and issue a FETCH statement followed by the INTO statements, print the results, check for exceptions and end the block.

Example numbers correspond to Section 4. For example example 4.15.3 is equivalent to 10.15.3. This is to allow you to look back and make comparisons. Some examples may be split here into two or more examples, those will have an additional number such as 10.15.3.1.

**Example 10.1.1.1**

To show all the records in the DEPARTMENT table when you're expecting multiple records to be returned, use the following DML:

```
-- Example 10.1.1.1

--      select *
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.1.1.1");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => DEPARTMENT ) );
```

# UNCLASSIFIED

```

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("DEPT_ID   DEPT_DESC");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT_ID );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- DEPT_ID
    PUT (V_DEPT_ID, 1);
    SET_COL (11); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

Now insert this into our EXAMPLES package framework and we have the following complete DML unit:

```

with TYPES, TABLES, VARIABLES;
  use TYPES, TABLES, VARIABLES;
with EXAMPLES_ADA_SQL;
  use EXAMPLES_ADA_SQL;
with TEXT_IO, CONVERSIONS;
  use CONVERSIONS;

procedure EXAMPLES is

  use TYPES.ADA_SQL;

  -- to do all the data conversions for displaying information

  use CONVERT_LAST_NAME, CONVERT_FIRST_NAME, CONVERT_DESCRIPTION_COURSE,
    CONVERT_GENERAL_ARRAY, I1_CONVERT, I2_CONVERT, I3_CONVERT, I4_CONVERT,

```

UNCLASSIFIED

```

I5_CONVERT, I6_CONVERT, CONVERT_FLOAT_YEARLY_INCOME,
CONVERT_FLOAT_TOTAL_INCOME, CONVERT_FLOAT_GRADE_POINT,
CONVERT_FLOAT_SALARY_RAISE, CONVERT_ENUMERATION_ENUMERATION_NUMBERS;

GOT_ONE : NATURAL := 0;

begin

    OPEN_DATABASE ("SYSTEM", "MANAGER");

    -- Example 10.1.1.1

    --      select *
    --      from DEPARTMENT ;

    NEW_LINE;
    PUT_LINE ("Output of Example 10.1.1.1");

    DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
            FROM => DEPARTMENT ) );

    OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("DEPT_ID  DEPT_DESC");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO ( V_DEPT_ID );
        INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- DEPT_ID
        PUT (V_DEPT_ID, 1);
        SET_COL (11); -- DEPT_DESC
        PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

## UNCLASSIFIED

```
CLOSE ( CURSOR );  
  
EXIT_DATABASE;  
  
end EXAMPLES;
```

This is the only time I will list the whole program. In the future all examples are assumed to be inserted into the framework. Run the Application Scanner against this program unit. If you have any errors look at the listing file, correct the errors and run the scanner again. Now compile the generated package then this package. Link the program and run it. Following should be your results:

### Output of Example 10.1.1.1

```
DEPT_ID  DEPT_DESC  
EXCEPTION: Not Found Error
```

The DBMS should tell you that the table currently has no records. We have created the tables but have not yet filled them with data.

### Example 10.1.1.2

To show all the records in the DEPARTMENT table when you're expecting only one record to be returned, use the following DML:

```
-- Example 10.1.1.2  
  
--      select *  
--      from DEPARTMENT ;  
  
NEW_LINE;  
PUT_LINE ("Output of Example 10.1.1.2");  
  
begin  
  
  SELEC ( '*',  
    FROM => DEPARTMENT );  
    INTO ( V_DEPT_ID );  
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );  
  
  NEW_LINE;  
  PUT_LINE ("DEPT_ID  DEPT_DESC");  
  SET_COL (1); -- DEPT_ID  
  PUT (V_DEPT_ID, 1);  
  SET_COL (11); -- DEPT_DESC  
  PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));  
  NEW_LINE;  
  
exception  
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
```



UNCLASSIFIED

```
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

Insert this into the framework, run the Application Scanner against this program unit. If you have any errors look at the listing file, correct the errors and run the scanner again. Now compile the generated package then this package. Link the program and run it. Following should be your results:

**Output of Example 10.1.1.2**

EXCEPTION: Not Found Error

**Example 10.1.2.1**

To show only one column of all the records in the DEPARTMENT table using the DECLAR cursor format you could use the statements:

```
-- Example 10.1.2.1

--      select DEPT_DESC
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.1.2.1");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( DEPT_DESC,
    FROM => DEPARTMENT ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ( "          DEPT_DESC" );
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (11); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
```

UNCLASSIFIED

```

                                else
                                    null;
                                end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

Run the program and the DBMS should tell you again that there is no data in the database.

**Output of Example 10.1.2.1**

```
DEPT_DESC
EXCEPTION: Not Found Error
```

**Example 10.1.2.2**

To show only one column of all the records in the DEPARTMENT table not using the DECLAR cursor format you could use the statements:

```
-- Example 10.1.2.2

--      select DEPT_DESC
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.1.2.2");

begin

    SELEC ( DEPT_DESC,
            FROM => DEPARTMENT );
            INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );

    NEW_LINE;
    PUT_LINE ( "          DEPT_DESC" );
    SET_COL (11); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    NEW_LINE;

exception
    when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

Run the program and the results are:

**Output of Example 10.1.2.2**

EXCEPTION: Not Found Error

**10.2 INSERT INTO, The Basics**

The next step is to put data into the tables we've created. This is done with the `INSERT INTO` statement. In this section we will discuss only the most simple form of the `INSERT INTO` statement. More complex forms will be discussed in a later chapter. To add a record to a table you must specify the table name and information for each column. The format of the `INSERT INTO` statement is:

```
INSERT INTO ( table ,
VALUES <= column_1_data and
          column_2_data and ... ) ;
```

You must supply data for every column in the table. Character strings must be enclosed in single quotes. Character string columns must be the maximum full length of the column. When a character string field won't fill up the column it should be padded with spaces. The "empty" characters in a character string must be ascii spaces when using Ada/SQL. Some DBMSs will automatically pad with spaces. Others will pad with a null value. If you are not sure how your DBMS will pad fill it with spaces yourself. All columns must be converted to the correct data type. Use Ada type conversion if necessary. We will be inserting column data at constants, it may be variables of the correct data type also. There is more discussion of `INSERT INTO` later on. This brief introduction to it is only for the purpose of getting data into our tables.

**Example 10.2.1**

Let's insert a record into the `DEPARTMENT` table. Ada/SQL allows only one record to be inserted for each `INSERT INTO` statement. Here's the code to do the insertion:

– Example 10.2.1

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.1");

INSERT INTO ( DEPARTMENT ,
VALUES <= TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          TYPES.ADA_SQL.DESCRPTION_DEPARTMENT'("History ") ) ;
```

And after scanning, compiling, linking and running we get:

**Output of Example 10.2.1**

Not much, but that's all we requested for output. Anything else would signify an error. If the `INSERT INTO` had not worked the program would have ended in an exception. From now on the program output will always follow the code without me specifically saying "and here's the output".

## UNCLASSIFIED

### Example 10.2.2

Now let's select all records and all fields from the DEPARTMENT table, using the same query we looked at earlier.

```
-- Example 10.2.2

--      select *
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.2.2");

begin
  SELEC ( '*',
    FROM => DEPARTMENT );
    INTO ( V_DEPT_ID );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );

  NEW_LINE;
  PUT_LINE ("DEPT_ID  DEPT_DESC");
  SET_COL (1); -- DEPT_ID
    PUT (V_DEPT_ID, 1);
  SET_COL (11); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
  NEW_LINE;

exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

### Output of Example 10.2.2

```
DEPT_ID  DEPT_DESC
      1  History
```

### Example 10.2.3

Now let's select only one field from all records from the DEPARTMENT table.

```
-- Example 10.2.3

--      select DEPT_DESC
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.2.3");
```

UNCLASSIFIED

```
begin
    SELEC ( DEPT_DESC,
        FROM => DEPARTMENT );
        INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );

    NEW_LINE;
    PUT_LINE ( "          DEPT_DESC" );
    SET_COL (11); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    NEW_LINE;

exception
    when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

**Output of Example 10.2.3**

```
DEPT_DESC
History
```

**Example 10.2.4 - 10.2.7**

We'll finish filling up the DEPARTMENT table with all the records we plan to have in it. Put the rest of the INSERT\_INTO statements one after the other in the program and we'll do them all together. Be sure to remove old query code from the framework before adding new queries or you'll be duplicating INSERT\_INTO statements and the database will end up with duplicate records.

**- Example 10.2.4**

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.4");

INSERT_INTO ( DEPARTMENT ,
VALUES <= TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
           TYPES.ADA_SQL.DESCRPTION_DEPARTMENT'("Math  ") ) ;
```

**-- Example 10.2.5**

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.5");

INSERT_INTO ( DEPARTMENT ,
VALUES <= TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
           TYPES.ADA_SQL.DESCRPTION_DEPARTMENT'("Science ") ) ;
```

## UNCLASSIFIED

-- Example 10.2.6

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.6");

INSERT INTO ( DEPARTMENT ,
VALUES <= TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.DESRIPTION_DEPARTMENT'("Language") ) ;
```

-- Example 10.2.7

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.7");

INSERT INTO ( DEPARTMENT ,
VALUES <= TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.DESRIPTION_DEPARTMENT'("Art      ") ) ;
```

**Output of Example 10.2.4**

**Output of Example 10.2.5**

**Output of Example 10.2.6**

**Output of Example 10.2.7**

### **Example 10.2.8**

Let's display all the records which have been inserted into the DEPARTMENT table. Your list of records may not be ordered exactly as this example is. The ordering of records in a relational database is insignificant. Later on we will discuss how to list records in a specified order.

-- Example 10.2.8

```
--      select *
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.2.8");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => DEPARTMENT ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
```

UNCLASSIFIED

```
PUT_LINE ("DEPT_ID  DEPT_DESC");
GOT_ONE := 0;

loop
  FETCH ( CURSOR );
  INTO ( V_DEPT_ID );
  INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- DEPT_ID
  PUT (V_DEPT_ID, 1);
  SET_COL (11); -- DEPT_DESC
  PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

**Output of Example 10.2.8**

DEPT_ID	DEPT_DESC
1	History
2	Math
3	Science
4	Language
5	Art

**Example 10.2.9 - 10.2.13**

The next several examples will fill the PROFESSOR table. Insert them all together in the frame work, the output for all is given at the end.

-- Example 10.2.9

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.9");

INSERT INTO ( PROFESSOR ,
VALUES <= TYPES.ADA_SQL.ID_PROFESSOR'(01) and
```

UNCLASSIFIED

```
TYPES.ADA_SQL.LAST_NAME'("Dysart      ") and
TYPES.ADA_SQL.FIRST_NAME'("Gregory    ") and
TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
TYPES.ADA_SQL.YEARS_EMPLOYED'(03) and
TYPES.ADA_SQL.YEARLY_INCOME'(35000.00) );
```

-- Example 10.2.10

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.10");

INSERT INTO ( PROFESSOR ,
VALUES <= TYPES.ADA_SQL.ID_PROFESSOR'(02) and
          TYPES.ADA_SQL.LAST_NAME'("Hall      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Elizabeth ") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(07) and
          TYPES.ADA_SQL.YEARLY_INCOME'(45000.00) );
```

-- Example 10.2.11

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.11");

INSERT INTO ( PROFESSOR ,
VALUES <= TYPES.ADA_SQL.ID_PROFESSOR'(03) and
          TYPES.ADA_SQL.LAST_NAME'("Steinbacner ") and
          TYPES.ADA_SQL.FIRST_NAME'("Moris      ") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(01) and
          TYPES.ADA_SQL.YEARLY_INCOME'(30000.00) );
```

-- Example 10.2.12

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.12");

INSERT INTO ( PROFESSOR ,
VALUES <= TYPES.ADA_SQL.ID_PROFESSOR'(04) and
          TYPES.ADA_SQL.LAST_NAME'("Bailey     ") and
          TYPES.ADA_SQL.FIRST_NAME'("Bruce      ") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(15) and
          TYPES.ADA_SQL.YEARLY_INCOME'(50000.00) );
```

-- Example 10.2.13

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.13");

INSERT INTO ( PROFESSOR ,
```



UNCLASSIFIED

```
VALUES <= TYPES.ADA_SQL.ID_PROFESSOR'(05) and
        TYPES.ADA_SQL.LAST_NAME'("Clements ") and
        TYPES.ADA_SQL.FIRST_NAME'("Carol ") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
        TYPES.ADA_SQL.YEARS_EMPLOYED'(04) and
        TYPES.ADA_SQL.YEARLY_INCOME'(40000.00) ) ;
```

Output of Example 10.2.9

Output of Example 10.2.10

Output of Example 10.2.11

Output of Example 10.2.12

Output of Example 10.2.13

Example 10.2.14

And now we'll take a look at the records in the PROFESSOR table.

-- Example 10.2.14

```
--      select *
--      from PROFESSOR ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.14");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => PROFESSOR ) );
```

```
OPEN ( CURSOR );
```

begin

```
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;
```

loop

```
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
```

# UNCLASSIFIED

```

GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.2.14

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

## Examples 10.2.15 - 10.2.30

We now fill the COURSE table with data.

-- Example 10.2.15

```

NEW_LINE;
PUT_LINE ("Output of Example 10.2.15");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(101) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(1) and

```

UNCLASSIFIED

```
TYPES.ADA_SQL.DESCRPTION_COURSE'("World History      ") and
TYPES.ADA_SQL.ID_PROFESSOR'(05) and
TWO ) ;
```

-- Example 10.2.16

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.16");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(102) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Political History      ") and
          TYPES.ADA_SQL.ID_PROFESSOR'(05) and
          THREE ) ;
```

-- Example 10.2.17

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.17");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(103) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Ancient History      ") and
          TYPES.ADA_SQL.ID_PROFESSOR'(05) and
          TWO ) ;
```

-- Example 10.2.18

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.18");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(201) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Algebra      ") and
          TYPES.ADA_SQL.ID_PROFESSOR'(03) and
          FOUR ) ;
```

-- Example 10.2.19

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.19");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(202) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Geometry      ") and
          TYPES.ADA_SQL.ID_PROFESSOR'(03) and
          FOUR ) ;
```

# UNCLASSIFIED

-- Example 10.2.20

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.20");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(203) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Trigonometry" ) and
          TYPES.ADA_SQL.ID_PROFESSOR'(03) and
          FIVE ) ;
```

-- Example 10.2.21

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.21");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(204) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Calculus" ) and
          TYPES.ADA_SQL.ID_PROFESSOR'(03) and
          FOUR ) ;
```

-- Example 10.2.22

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.22");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(301) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Chemistry" ) and
          TYPES.ADA_SQL.ID_PROFESSOR'(01) and
          THREE ) ;
```

-- Example 10.2.23

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.23");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(302) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Physics" ) and
          TYPES.ADA_SQL.ID_PROFESSOR'(01) and
          FIVE ) ;
```

-- Example 10.2.24

```
NEW_LINE;
```

# UNCLASSIFIED

PUT\_LINE ("Output of Example 10.2.24");

```
INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(303) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Biology" and
          TYPES.ADA_SQL.ID_PROFESSOR'(01) and
          FOUR ) ;
```

-- Example 10.2.25

NEW\_LINE;  
PUT\_LINE ("Output of Example 10.2.25");

```
INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(401) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("French" and
          TYPES.ADA_SQL.ID_PROFESSOR'(02) and
          TWO ) ;
```

-- Example 10.2.26

NEW\_LINE;  
PUT\_LINE ("Output of Example 10.2.26");

```
INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(402) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Spanish" and
          TYPES.ADA_SQL.ID_PROFESSOR'(05) and
          TWO ) ;
```

-- Example 10.2.27

NEW\_LINE;  
PUT\_LINE ("Output of Example 10.2.27");

```
INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(403) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Russian" and
          TYPES.ADA_SQL.ID_PROFESSOR'(02) and
          FOUR ) ;
```

-- Example 10.2.28

NEW\_LINE;  
PUT\_LINE ("Output of Example 10.2.28");

INSERT INTO ( COURSE ,

UNCLASSIFIED

```
VALUES <= TYPES.ADA_SQL.ID_COURSE'(501) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Sculpture" and
          TYPES.ADA_SQL.ID_PROFESSOR'(04) and
          ONE ) ;
```

-- Example 10.2.29

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.29");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(502) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Music" and
          TYPES.ADA_SQL.ID_PROFESSOR'(04) and
          ONE ) ;
```

-- Example 10.2.30

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.30");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(503) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Dance" and
          TYPES.ADA_SQL.ID_PROFESSOR'(05) and
          TWO ) ;
```

Output of Example 10.2.15

Output of Example 10.2.16

Output of Example 10.2.17

Output of Example 10.2.18

Output of Example 10.2.19

Output of Example 10.2.20

Output of Example 10.2.21

Output of Example 10.2.22

Output of Example 10.2.23

Output of Example 10.2.24

Output of Example 10.2.25

UNCLASSIFIED

Output of Example 10.2.26

Output of Example 10.2.27

Output of Example 10.2.28

Output of Example 10.2.29

Output of Example 10.2.30

Example 10.2.31

List all the records currently in the COURSE table.

-- Example 10.2.31

```
--      select *
--      from COURSE ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.31");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => COURSE ) );
```

```
OPEN ( CURSOR );
```

begin

```
NEW_LINE;
PUT_LINE ("COURSE_ID  COURSE_DEPT  COURSE_DESC           " &
        "COURSE_PROF  COURSE_HOURS");
GOT_ONE := 0;
```

loop

```
  FETCH ( CURSOR );
  INTO (V_COURSE_ID);
  INTO (V_COURSE_DEPT);
  INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
  INTO (V_COURSE_PROF);
  INTO (V_COURSE_HOURS);
  GOT_ONE := GOT_ONE + 1;
```

```
  SET_COL (1); -- COURSE_ID
  PUT (V_COURSE_ID, 3);
  SET_COL (12); -- COURSE_DEPT
  PUT (V_COURSE_DEPT, 1);
  SET_COL (24); -- COURSE_DESC
  PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
  SET_COL (46); -- COURSE_PROF
```

# UNCLASSIFIED

```

        PUT (V_COURSE_PROF, 2);
        SET_COL (58); -- COURSE_HOURS
        PUT (V_COURSE_HOURS);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.2.31

COURSE_ID	COURSE_DEPT	COURSE_DESC	COURSE_PROF	COURSE_HOURS
101	1	World History	5	TWO
102	1	Political History	5	THREE
103	1	Ancient History	5	TWO
201	2	Algebra	3	FOUR
202	2	Geometry	3	FOUR
203	2	Trigonometry	3	FIVE
204	2	Calculus	3	FOUR
301	3	Chemistry	1	THREE
302	3	Physics	1	FIVE
303	3	Biology	1	FOUR
401	4	French	2	TWO
402	4	Spanish	5	TWO
403	4	Russian	2	FOUR
501	5	Sculpture	4	ONE
502	5	Music	4	ONE
503	5	Dance	5	TWO

## Example 10.2.32

And now fill up the STUDENT table with data.

-- Example 10.2.32

```

NEW_LINE;
PUT_LINE ("Output of Example 10.2.32");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT' (001) and
        TYPES.ADA_SQL.LAST_NAME' ("Horrigan ") and

```



UNCLASSIFIED

```
TYPES.ADA_SQL.FIRST_NAME'("William  ") and
TYPES.ADA_SQL.GENERAL_ARRAY '("A101") and
TYPES.ADA_SQL.HOME_STATE'("VA") and
TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
FOUR ) ;
```

-- Example 10.2.33

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.33");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(002) and
          TYPES.ADA_SQL.LAST_NAME'("McGinn      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Gregory    ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A102") and
          TYPES.ADA_SQL.HOME_STATE'("MD") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          THREE ) ;
```

-- Example 10.2.34

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.34");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(003) and
          TYPES.ADA_SQL.LAST_NAME'("Lewis      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Molly     ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A103") and
          TYPES.ADA_SQL.HOME_STATE'("PA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TWO ) ;
```

-- Example 10.2.35

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.35");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(004) and
          TYPES.ADA_SQL.LAST_NAME'("Waxler     ") and
          TYPES.ADA_SQL.FIRST_NAME'("Dennis    ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A104") and
          TYPES.ADA_SQL.HOME_STATE'("NC") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TWO ) ;
```

-- Example 10.2.36

```
NEW_LINE;
```

UNCLASSIFIED

```
PUT_LINE ("Output of Example 10.2.36");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(005) and
          TYPES.ADA_SQL.LAST_NAME'("McNamara      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Howard      ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A201") and
          TYPES.ADA_SQL.HOME_STATE'("VA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          ONE ) ;
```

-- Example 10.2.37

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.37");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(006) and
          TYPES.ADA_SQL.LAST_NAME'("Hess          ") and
          TYPES.ADA_SQL.FIRST_NAME'("Fay          ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A202") and
          TYPES.ADA_SQL.HOME_STATE'("DC") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          THREE ) ;
```

-- Example 10.2.38

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.38");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(007) and
          TYPES.ADA_SQL.LAST_NAME'("Guiffre       ") and
          TYPES.ADA_SQL.FIRST_NAME'("Jennifer   ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A203") and
          TYPES.ADA_SQL.HOME_STATE'("MD") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          ONE ) ;
```

-- Example 10.2.39

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.39");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(008) and
          TYPES.ADA_SQL.LAST_NAME'("Hagan         ") and
          TYPES.ADA_SQL.FIRST_NAME'("Carl         ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A204") and
          TYPES.ADA_SQL.HOME_STATE'("PA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
```

UNCLASSIFIED

FOUR ) ;

-- Example 10.2.40

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.40");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(009) and
          TYPES.ADA_SQL.LAST_NAME'("Bearman    ") and
          TYPES.ADA_SQL.FIRST_NAME'("Rose      ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A301") and
          TYPES.ADA_SQL.HOME_STATE'("VA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          ONE ) ;
```

-- Example 10.2.41

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.41");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(010) and
          TYPES.ADA_SQL.LAST_NAME'("Thompson   ") and
          TYPES.ADA_SQL.FIRST_NAME'("Paul       ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A302") and
          TYPES.ADA_SQL.HOME_STATE'("NC") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          THREE ) ;
```

-- Example 10.2.42

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.42");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(011) and
          TYPES.ADA_SQL.LAST_NAME'("Bennett     ") and
          TYPES.ADA_SQL.FIRST_NAME'("Nellie    ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A303") and
          TYPES.ADA_SQL.HOME_STATE'("PA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          THREE ) ;
```

-- Example 10.2.43

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.43");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(012) and
```

UNCLASSIFIED

```
TYPES.ADA_SQL.LAST_NAME'("Schmidt    ") and
TYPES.ADA_SQL.FIRST_NAME'("John      ") and
TYPES.ADA_SQL.GENERAL_ARRAY'("A304") and
TYPES.ADA_SQL.HOME_STATE'("SC") and
TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
TWO ) ;
```

-- Example 10.2.44

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.44");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(013) and
          TYPES.ADA_SQL.LAST_NAME'("Gevarter    ") and
          TYPES.ADA_SQL.FIRST_NAME'("Susan      ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("B101") and
          TYPES.ADA_SQL.HOME_STATE'("NY") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          FOUR ) ;
```

-- Example 10.2.45

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.45");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(014) and
          TYPES.ADA_SQL.LAST_NAME'("Sherman     ") and
          TYPES.ADA_SQL.FIRST_NAME'("Donald    ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("B102") and
          TYPES.ADA_SQL.HOME_STATE'("VA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          THREE ) ;
```

-- Example 10.2.46

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.46");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(015) and
          TYPES.ADA_SQL.LAST_NAME'("Gorham      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Milton     ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("B103") and
          TYPES.ADA_SQL.HOME_STATE'("WV") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TWO ) ;
```

-- Example 10.2.47

# UNCLASSIFIED

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.47");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and
          TYPES.ADA_SQL.LAST_NAME'("Williams    ") and
          TYPES.ADA_SQL.FIRST_NAME'("Alvin      ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("B104") and
          TYPES.ADA_SQL.HOME_STATE'("DC") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          ONE ) ;
```

-- Example 10.2.48

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.48");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(017) and
          TYPES.ADA_SQL.LAST_NAME'("Woodliff    ") and
          TYPES.ADA_SQL.FIRST_NAME'("Dorothy    ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("B201") and
          TYPES.ADA_SQL.HOME_STATE'("MD") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          FOUR ) ;
```

-- Example 10.2.49

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.49");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(018) and
          TYPES.ADA_SQL.LAST_NAME'("Ratliff     ") and
          TYPES.ADA_SQL.FIRST_NAME'("Ann        ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("B202") and
          TYPES.ADA_SQL.HOME_STATE'("NY") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          ONE ) ;
```

-- Example 10.2.50

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.50");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(019) and
          TYPES.ADA_SQL.LAST_NAME'("Phung       ") and
          TYPES.ADA_SQL.FIRST_NAME'("Kim        ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("B203") and
          TYPES.ADA_SQL.HOME_STATE'("SC") and
```

UNCLASSIFIED

```
TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
TWO ) ;
```

-- Example 10.2.51

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.51");  
  
INSERT INTO ( STUDENT ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(020) and  
          TYPES.ADA_SQL.LAST_NAME'("McMurray ") and  
          TYPES.ADA_SQL.FIRST_NAME'("Eric ") and  
          TYPES.ADA_SQL.GENERAL_ARRAY '("B204") and  
          TYPES.ADA_SQL.HOME_STATE'("VA") and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
          ONE ) ;
```

-- Example 10.2.52

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.52");  
  
INSERT INTO ( STUDENT ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(021) and  
          TYPES.ADA_SQL.LAST_NAME'("O'Leary ") and  
          TYPES.ADA_SQL.FIRST_NAME'("Peggy ") and  
          TYPES.ADA_SQL.GENERAL_ARRAY '("C101") and  
          TYPES.ADA_SQL.HOME_STATE'("PA") and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and  
          FOUR ) ;
```

-- Example 10.2.53

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.53");  
  
INSERT INTO ( STUDENT ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(022) and  
          TYPES.ADA_SQL.LAST_NAME'("Martin ") and  
          TYPES.ADA_SQL.FIRST_NAME'("Charoltte ") and  
          TYPES.ADA_SQL.GENERAL_ARRAY '("C102") and  
          TYPES.ADA_SQL.HOME_STATE'("DC") and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and  
          TWO ) ;
```

-- Example 10.2.54

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.54");  
  
INSERT INTO ( STUDENT ,
```

UNCLASSIFIED

```
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(023) and
          TYPES.ADA_SQL.LAST_NAME'("O'Day      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Hilda      ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("C103") and
          TYPES.ADA_SQL.HOME_STATE'("NC") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          ONE ) ;
```

-- Example 10.2.55

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.55");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(024) and
          TYPES.ADA_SQL.LAST_NAME'("Martin      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Edward      ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("C104") and
          TYPES.ADA_SQL.HOME_STATE'("MD") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          THREE ) ;
```

-- Example 10.2.56

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.56");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(025) and
          TYPES.ADA_SQL.LAST_NAME'("Chateaufneuf ") and
          TYPES.ADA_SQL.FIRST_NAME'("Chelsea    ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("C105") and
          TYPES.ADA_SQL.HOME_STATE'("VA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          THREE ) ;
```

Output of Example 10.2.32

Output of Example 10.2.33

Output of Example 10.2.34

Output of Example 10.2.35

Output of Example 10.2.36

Output of Example 10.2.37

Output of Example 10.2.38

Output of Example 10.2.39

UNCLASSIFIED

Output of Example 10.2.40

Output of Example 10.2.41

Output of Example 10.2.42

Output of Example 10.2.43

Output of Example 10.2.44

Output of Example 10.2.45

Output of Example 10.2.46

Output of Example 10.2.47

Output of Example 10.2.48

Output of Example 10.2.49

Output of Example 10.2.50

Output of Example 10.2.51

Output of Example 10.2.52

Output of Example 10.2.53

Output of Example 10.2.54

Output of Example 10.2.55

Output of Example 10.2.56

Example 10.2.57

List all the records stored in the STUDENT table.

-- Example 10.2.57

```
--      select *  
--      from STUDENT ;
```

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.57");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>  
        SELEC ( '*' ,  
              FROM => STUDENT ) );
```



# UNCLASSIFIED

```

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
       "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;

  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.2.57

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
-------	---------	----------	---------	----------	----------	---------

# UNCLASSIFIED

1	Horrigan	William	A101	VA	3	FOUR
2	McGinn	Gregory	A102	MD	1	THREE
3	Lewis	Molly	A103	PA	4	TWO
4	Waxler	Dennis	A104	NC	2	TWO
5	McNamara	Howard	A201	VA	5	ONE
6	Hess	Fay	A202	DC	3	THREE
7	Guiffre	Jennifer	A203	MD	4	ONE
8	Hagan	Carl	A204	PA	5	FOUR
9	Bearman	Rose	A301	VA	2	ONE
10	Thompson	Paul	A302	NC	1	THREE
11	Bennett	Nellie	A303	PA	4	THREE
12	Schmidt	John	A304	SC	5	TWO
13	Gevarter	Susan	B101	NY	5	FOUR
14	Sherman	Donald	B102	VA	3	THREE
15	Gorham	Milton	B103	WV	2	TWO
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR
18	Ratliff	Ann	B202	NY	5	ONE
19	Phung	Kim	B203	SC	2	TWO
20	McMurray	Eric	B204	VA	2	ONE
21	O'Leary	Peggy	C101	PA	3	FOUR
22	Martin	Charlotte	C102	DC	1	TWO
23	O'Day	Hilda	C103	NC	4	ONE
24	Martin	Edward	C104	MD	5	THREE
25	Chateaufneuf	Chelsea	C105	VA	1	THREE

## Example 10.2.58

Now fill up the CLASS table with information.

-- Example 10.2.58

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.58");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(001) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.ID_COURSE'(302) and
          TYPES.ADA_SQL.GRADE_POINT'(089.49) and
          TYPES.ADA_SQL.GRADE_POINT'(051.91) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.59

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.59");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(001) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.ID_COURSE'(303) and
```

UNCLASSIFIED

```
TYPES.ADA_SQL.GRADE_POINT'(077.61) and  
TYPES.ADA_SQL.GRADE_POINT'(088.84) and  
TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.60

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.60");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(002) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and  
          TYPES.ADA_SQL.ID_COURSE'(103) and  
          TYPES.ADA_SQL.GRADE_POINT'(054.38) and  
          TYPES.ADA_SQL.GRADE_POINT'(084.77) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.61

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.61");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(003) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          TYPES.ADA_SQL.ID_COURSE'(403) and  
          TYPES.ADA_SQL.GRADE_POINT'(092.92) and  
          TYPES.ADA_SQL.GRADE_POINT'(097.48) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.62

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.62");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(004) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
          TYPES.ADA_SQL.ID_COURSE'(204) and  
          TYPES.ADA_SQL.GRADE_POINT'(071.17) and  
          TYPES.ADA_SQL.GRADE_POINT'(070.55) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.63

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.63");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(005) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
```

UNCLASSIFIED

```
TYPES.ADA_SQL.ID_COURSE'(503) and  
TYPES.ADA_SQL.GRADE_POINT'(088.83) and  
TYPES.ADA_SQL.GRADE_POINT'(081.12) and  
TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.64

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.64");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(006) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and  
          TYPES.ADA_SQL.ID_COURSE'(301) and  
          TYPES.ADA_SQL.GRADE_POINT'(066.26) and  
          TYPES.ADA_SQL.GRADE_POINT'(094.60) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.65

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.65");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(006) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          TYPES.ADA_SQL.ID_COURSE'(402) and  
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and  
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.66

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.66");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(007) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          TYPES.ADA_SQL.ID_COURSE'(401) and  
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and  
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.67

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.67");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(007) and
```

UNCLASSIFIED

```
TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
TYPES.ADA_SQL.ID_COURSE'(402) and
TYPES.ADA_SQL.GRADE_POINT'(100.00) and
TYPES.ADA_SQL.GRADE_POINT'(100.00) and
TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.68

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.68");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(007) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.ID_COURSE'(403) and
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.69

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.69");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(007) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.ID_COURSE'(503) and
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.70

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.70");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(008) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.ID_COURSE'(502) and
          TYPES.ADA_SQL.GRADE_POINT'(069.68) and
          TYPES.ADA_SQL.GRADE_POINT'(056.92) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.71

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.71");

INSERT INTO ( CLASS ,
```

UNCLASSIFIED

```
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(009) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
        TYPES.ADA_SQL.ID_COURSE'(204) and
        TYPES.ADA_SQL.GRADE_POINT'(055.53) and
        TYPES.ADA_SQL.GRADE_POINT'(089.81) and
        TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.72

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.72");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(010) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
        TYPES.ADA_SQL.ID_COURSE'(102) and
        TYPES.ADA_SQL.GRADE_POINT'(093.72) and
        TYPES.ADA_SQL.GRADE_POINT'(099.55) and
        TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.73

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.73");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(011) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
        TYPES.ADA_SQL.ID_COURSE'(401) and
        TYPES.ADA_SQL.GRADE_POINT'(081.99) and
        TYPES.ADA_SQL.GRADE_POINT'(076.29) and
        TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.74

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.74");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(012) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
        TYPES.ADA_SQL.ID_COURSE'(501) and
        TYPES.ADA_SQL.GRADE_POINT'(075.81) and
        TYPES.ADA_SQL.GRADE_POINT'(083.03) and
        TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.75

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.75");
```

UNCLASSIFIED

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(013) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(502) and  
          TYPES.ADA_SQL.GRADE_POINT'(067.36) and  
          TYPES.ADA_SQL.GRADE_POINT'(080.15) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.76

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.76");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(014) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and  
          TYPES.ADA_SQL.ID_COURSE'(302) and  
          TYPES.ADA_SQL.GRADE_POINT'(092.27) and  
          TYPES.ADA_SQL.GRADE_POINT'(082.47) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.77

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.77");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(015) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
          TYPES.ADA_SQL.ID_COURSE'(202) and  
          TYPES.ADA_SQL.GRADE_POINT'(089.75) and  
          TYPES.ADA_SQL.GRADE_POINT'(095.74) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.78

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.78");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and  
          TYPES.ADA_SQL.ID_COURSE'(101) and  
          TYPES.ADA_SQL.GRADE_POINT'(085.64) and  
          TYPES.ADA_SQL.GRADE_POINT'(078.26) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.79

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.79");
```

UNCLASSIFIED

```
INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          TYPES.ADA_SQL.ID_COURSE'(101) and
          TYPES.ADA_SQL.GRADE_POINT'(094.59) and
          TYPES.ADA_SQL.GRADE_POINT'(091.52) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.80

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.80");
```

```
INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.ID_COURSE'(204) and
          TYPES.ADA_SQL.GRADE_POINT'(083.40) and
          TYPES.ADA_SQL.GRADE_POINT'(094.88) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.81

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.81");
```

```
INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.ID_COURSE'(302) and
          TYPES.ADA_SQL.GRADE_POINT'(082.14) and
          TYPES.ADA_SQL.GRADE_POINT'(087.11) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.82

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.82");
```

```
INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.ID_COURSE'(403) and
          TYPES.ADA_SQL.GRADE_POINT'(089.92) and
          TYPES.ADA_SQL.GRADE_POINT'(097.40) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.83

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.83");
```



UNCLASSIFIED

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(501) and  
          TYPES.ADA_SQL.GRADE_POINT'(076.86) and  
          TYPES.ADA_SQL.GRADE_POINT'(095.72) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.84

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.84");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(017) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          TYPES.ADA_SQL.ID_COURSE'(401) and  
          TYPES.ADA_SQL.GRADE_POINT'(094.71) and  
          TYPES.ADA_SQL.GRADE_POINT'(063.36) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.85

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.85");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(018) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(503) and  
          TYPES.ADA_SQL.GRADE_POINT'(092.69) and  
          TYPES.ADA_SQL.GRADE_POINT'(071.69) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.86

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.86");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(019) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
          TYPES.ADA_SQL.ID_COURSE'(201) and  
          TYPES.ADA_SQL.GRADE_POINT'(081.31) and  
          TYPES.ADA_SQL.GRADE_POINT'(095.95) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.87

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.87");
```

UNCLASSIFIED

```
INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(020) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.ID_COURSE'(204) and
          TYPES.ADA_SQL.GRADE_POINT'(088.28) and
          TYPES.ADA_SQL.GRADE_POINT'(079.01) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;

-- Example 10.2.88

NEW_LINE;
PUT_LINE ("Output of Example 10.2.88");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(021) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.ID_COURSE'(303) and
          TYPES.ADA_SQL.GRADE_POINT'(071.16) and
          TYPES.ADA_SQL.GRADE_POINT'(074.14) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;

-- Example 10.2.89

NEW_LINE;
PUT_LINE ("Output of Example 10.2.89");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(022) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          TYPES.ADA_SQL.ID_COURSE'(102) and
          TYPES.ADA_SQL.GRADE_POINT'(058.97) and
          TYPES.ADA_SQL.GRADE_POINT'(086.58) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;

-- Example 10.2.90

NEW_LINE;
PUT_LINE ("Output of Example 10.2.90");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(022) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.ID_COURSE'(201) and
          TYPES.ADA_SQL.GRADE_POINT'(081.75) and
          TYPES.ADA_SQL.GRADE_POINT'(092.97) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;

-- Example 10.2.91

NEW_LINE;
PUT_LINE ("Output of Example 10.2.91");
```

UNCLASSIFIED

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(022) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(503) and  
          TYPES.ADA_SQL.GRADE_POINT'(074.49) and  
          TYPES.ADA_SQL.GRADE_POINT'(098.30) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.92

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.92");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(023) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          TYPES.ADA_SQL.ID_COURSE'(402) and  
          TYPES.ADA_SQL.GRADE_POINT'(096.33) and  
          TYPES.ADA_SQL.GRADE_POINT'(081.53) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.93

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.93");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(024) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(503) and  
          TYPES.ADA_SQL.GRADE_POINT'(097.14) and  
          TYPES.ADA_SQL.GRADE_POINT'(085.72) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.94

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.94");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(025) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and  
          TYPES.ADA_SQL.ID_COURSE'(101) and  
          TYPES.ADA_SQL.GRADE_POINT'(083.58) and  
          TYPES.ADA_SQL.GRADE_POINT'(089.16) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

Output of Example 10.2.58

Output of Example 10.2.59

**UNCLASSIFIED**

**Output of Example 10.2.60**

**Output of Example 10.2.61**

**Output of Example 10.2.62**

**Output of Example 10.2.63**

**Output of Example 10.2.64**

**Output of Example 10.2.65**

**Output of Example 10.2.66**

**Output of Example 10.2.67**

**Output of Example 10.2.68**

**Output of Example 10.2.69**

**Output of Example 10.2.70**

**Output of Example 10.2.71**

**Output of Example 10.2.72**

**Output of Example 10.2.73**

**Output of Example 10.2.74**

**Output of Example 10.2.75**

**Output of Example 10.2.76**

**Output of Example 10.2.77**

**Output of Example 10.2.78**

**Output of Example 10.2.79**

**Output of Example 10.2.80**

**Output of Example 10.2.81**

**Output of Example 10.2.82**

**Output of Example 10.2.83**

**Output of Example 10.2.84**

**Output of Example 10.2.85**

UNCLASSIFIED

Output of Example 10.2.86

Output of Example 10.2.87

Output of Example 10.2.88

Output of Example 10.2.89

Output of Example 10.2.90

Output of Example 10.2.91

Output of Example 10.2.92

Output of Example 10.2.93

Output of Example 10.2.94

Example 10.2.95

And let's take a look at the data we inserted into the CLASS table.

-- Example 10.2.95

```
--      select *  
--      from CLASS ;
```

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.95");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>  
        SELEC ( '*',  
              FROM => CLASS) );
```

```
OPEN ( CURSOR );
```

begin

```
NEW_LINE;  
PUT_LINE ("CLASS_STUDENT CLASS_DEPT CLASS_COURSE CLASS_SEM_1 " &  
        "CLASS_SEM_2 CLASS_GRADE");  
GOT_ONE := 0;
```

loop

```
    FETCH ( CURSOR );  
    INTO (V_CLASS_STUDENT);  
    INTO (V_CLASS_DEPT);  
    INTO (V_CLASS_COURSE);  
    INTO (V_CLASS_SEM_1);  
    INTO (V_CLASS_SEM_2);  
    INTO (V_CLASS_GRADE);
```

# UNCLASSIFIED

```

GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- CLASS_STUDENT
PUT (V_CLASS_STUDENT, 3);
SET_COL (15); -- CLASS_DEPT
PUT (V_CLASS_DEPT, 1);
SET_COL (26); -- CLASS_COURSE
PUT (V_CLASS_COURSE, 3);
SET_COL (39); -- CLASS_SEM_1
PUT (V_CLASS_SEM_1, 3, 2, 0);
SET_COL (51); -- CLASS_SEM_2
PUT (V_CLASS_SEM_2, 3, 2, 0);
SET_COL (63); -- CLASS_GRADE
PUT (V_CLASS_GRADE, 3, 2, 0);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.2.95

CLASS_STUDENT	CLASS_DEPT	CLASS_COURSE	CLASS_SEM_1	CLASS_SEM_2	CLASS_GRADE
1	3	302	89.49	51.91	0.00
1	3	303	77.61	88.84	0.00
2	1	103	54.38	84.77	0.00
3	4	403	92.92	97.48	0.00
4	2	204	71.17	70.55	0.00
5	5	503	88.83	81.12	0.00
6	3	301	66.26	94.60	0.00
6	4	402	100.00	100.00	0.00
7	4	401	100.00	100.00	0.00
7	4	402	100.00	100.00	0.00
7	4	403	100.00	100.00	0.00
7	5	503	100.00	100.00	0.00
8	5	502	69.68	56.92	0.00
9	2	204	55.53	89.81	0.00
10	1	102	93.72	99.55	0.00
11	4	401	81.99	76.29	0.00
12	5	501	75.81	83.03	0.00
13	5	502	67.36	80.15	0.00

# UNCLASSIFIED

14	3	302	92.27	82.47	0.00
15	2	202	89.75	95.74	0.00
16	1	101	85.64	78.26	0.00
16	1	101	94.59	91.52	0.00
16	2	204	83.40	94.88	0.00
16	3	302	82.14	87.11	0.00
16	4	403	89.92	97.40	0.00
16	5	501	76.86	95.72	0.00
17	4	401	94.71	63.36	0.00
18	5	503	92.69	71.69	0.00
19	2	201	81.31	95.95	0.00
20	2	204	88.28	79.01	0.00
21	3	303	71.16	74.14	0.00
22	1	102	58.97	86.58	0.00
22	2	201	81.75	92.97	0.00
22	5	503	74.49	98.30	0.00
23	4	402	96.33	81.53	0.00
24	5	503	97.14	85.72	0.00
25	1	101	83.58	89.16	0.00

## Example 10.2.96

Now we'll fill the SALARY table with data.

-- Example 10.2.96

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.96");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(1) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(1) and
          TYPES.ADA_SQL.YEARLY_INCOME'(20000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(29999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.010) ) ;
```

-- Example 10.2.97

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.97");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(2) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(2) and
          TYPES.ADA_SQL.YEARLY_INCOME'(30000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(34999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.075) ) ;
```

-- Example 10.2.98

```
NEW_LINE;
```

UNCLASSIFIED

```
PUT_LINE ("Output of Example 10.2.98");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(3) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(3) and
          TYPES.ADA_SQL.YEARLY_INCOME'(35000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(39999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.050) ) ;

-- Example 10.2.99

NEW_LINE;
PUT_LINE ("Output of Example 10.2.99");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(4) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(4) and
          TYPES.ADA_SQL.YEARLY_INCOME'(40000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(44999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.035) ) ;

-- Example 10.2.100

NEW_LINE;
PUT_LINE ("Output of Example 10.2.100");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(5) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(5) and
          TYPES.ADA_SQL.YEARLY_INCOME'(45000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(49999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.025) ) ;

-- Example 10.2.101

NEW_LINE;
PUT_LINE ("Output of Example 10.2.101");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(6) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(10) and
          TYPES.ADA_SQL.YEARLY_INCOME'(50000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(51999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.020) ) ;

-- Example 10.2.102

NEW_LINE;
PUT_LINE ("Output of Example 10.2.102");

INSERT INTO ( SALARY ,
```



UNCLASSIFIED

```
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(11) and
        TYPES.ADA_SQL.YEARS_EMPLOYED'(15) and
        TYPES.ADA_SQL.YEARLY_INCOME'(52000.00) and
        TYPES.ADA_SQL.YEARLY_INCOME'(53999.00) and
        TYPES.ADA_SQL.SALARY_RAISE'(0.020) ) ;
```

-- Example 10.2.103

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.103");
```

```
INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(16) and
        TYPES.ADA_SQL.YEARS_EMPLOYED'(20) and
        TYPES.ADA_SQL.YEARLY_INCOME'(54000.00) and
        TYPES.ADA_SQL.YEARLY_INCOME'(55999.00) and
        TYPES.ADA_SQL.SALARY_RAISE'(0.020) ) ;
```

-- Example 10.2.104

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.104");
```

```
INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(21) and
        TYPES.ADA_SQL.YEARS_EMPLOYED'(99) and
        TYPES.ADA_SQL.YEARLY_INCOME'(56000.00) and
        TYPES.ADA_SQL.YEARLY_INCOME'(60000.00) and
        TYPES.ADA_SQL.SALARY_RAISE'(0.020) ) ;
```

Output of Example 10.2.96

Output of Example 10.2.97

Output of Example 10.2.98

Output of Example 10.2.99

Output of Example 10.2.100

Output of Example 10.2.101

Output of Example 10.2.102

Output of Example 10.2.103

Output of Example 10.2.104

Example 10.2.105

# UNCLASSIFIED

And let's take a look at the information in the SALARY table.

-- Example 10.2.105

```
--      select *
--      from SALARY ;

NEW_LINE;
PUT_LINE ("Output of Example 10.2.105");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => SALARY ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("SAL_YEAR  SAL_END  SAL_MIN  SAL_MAX  SAL_RAISE");
GOT_ONE := 0;

loop
  FETCH ( CURSOR );
  INTO (V_SAL_YEAR);
  INTO (V_SAL_END);
  INTO (V_SAL_MIN);
  INTO (V_SAL_MAX);
  INTO (V_SAL_RAISE);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- SAL_YEAR
  PUT (V_SAL_YEAR, 2);
  SET_COL (11); -- SAL_END
  PUT (V_SAL_END, 2);
  SET_COL (20); -- SAL_MIN
  PUT (V_SAL_MIN, 5, 2, 0);
  SET_COL (30); -- SAL_MAX
  PUT (V_SAL_MAX, 5, 2, 0);
  SET_COL (40); -- SAL_RAISE
  PUT (V_SAL_RAISE, 1, 3, 0);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                          PUT_LINE ("EXCEPTION: Not Found Error");
                          else
                            null;
                          end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
```

UNCLASSIFIED

end;

CLOSE ( CURSOR );

**Output of Example 10.2.105**

SAL_YEAR	SAL_END	SAL_MIN	SAL_MAX	SAL_RAISE
1	1	20000.00	29999.00	0.010
2	2	30000.00	34999.00	0.075
3	3	35000.00	39999.00	0.050
4	4	40000.00	44999.00	0.035
5	5	45000.00	49999.00	0.025
6	10	50000.00	51999.00	0.020
11	15	52000.00	53999.00	0.020
16	20	54000.00	55999.00	0.020
21	99	56000.00	60000.00	0.020

We'll leave the GRADE table empty for now.

**10.3 More Basic SELECTS**

Before learning more query commands lets try a couple more selections of data.

**Example 10.3.1**

List all students, their first names, last names, room number and year.

-- Example 10.3.1

```
--      select ST_FIRST, ST_NAME, ST_ROOM, ST_YEAR
--      from STUDENT ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.3.1");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_FIRST & ST_NAME & ST_ROOM & ST_YEAR,
        FROM => STUDENT ) );
```

```
OPEN ( CURSOR );
```

```
begin
```

```
NEW_LINE;
PUT ("ST_FIRST      ST_NAME      ST_ROOM  ST_YEAR");
GOT_ONE := 0;
```

```
loop
```

```
  FETCH ( CURSOR );
  INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
```

# UNCLASSIFIED

```

    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (13); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (27); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (36); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.3.1

ST_FIRST	ST_NAME	ST_ROOM	ST_YEAR
William	Horrigan	A101	FOUR
Gregory	McGinn	A102	THREE
Molly	Lewis	A103	TWO
Dennis	Waxler	A104	TWO
Howard	McNamara	A201	ONE
Fay	Hess	A202	THREE
Jennifer	Guiffre	A203	ONE
Carl	Hagan	A204	FOUR
Rose	Bearman	A301	ONE
Paul	Thompson	A302	THREE
Nellie	Bennett	A303	THREE
John	Schmidt	A304	TWO
Susan	Gevarter	B101	FOUR
Donald	Sherman	B102	THREE
Milton	Gorham	B103	TWO
Alvin	Williams	B104	ONE
Dorothy	Woodliff	B201	FOUR
Ann	Ratliff	B202	ONE
Kim	Phung	B203	TWO

# UNCLASSIFIED

Eric	McMurray	B204	ONE
Peggy	O'Leary	C101	FOUR
Charlotte	Martin	C102	TWO
Hilda	O'Day	C103	ONE
Edward	Martin	C104	THREE
Chelsea	Chateauneuf	C105	THREE

## Example 10.3.2

Now give me a list of all the professors, their last names, salaries and number of years at the univers

-- Example 10.3.2

```
--      select PROF_NAME, PROF_SALARY, PROF_YEARS
--      from PROFESSOR ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.3.2");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_NAME & PROF_SALARY & PROF_YEARS,
                FROM => PROFESSOR ) );
```

```
OPEN ( CURSOR );
```

begin

```
NEW_LINE;
PUT_LINE ("PROF_NAME      PROF_SALARY  PROF_YEARS");
GOT_ONE := 0;
```

loop

```
FETCH ( CURSOR );
INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
INTO ( V_PROF_SALARY );
INTO ( V_PROF_YEARS );
GOT_ONE := GOT_ONE + 1;
```

```
SET_COL (1); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (15); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
SET_COL (28); -- PROF_YEARS
PUT (V_PROF_YEARS, 2);
```

```
NEW_LINE;
end loop;
```

exception

```
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                        PUT_LINE ("EXCEPTION: Not Found Error");
                        else
```

## UNCLASSIFIED

```
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.3.2

PROF_NAME	PROF_SALARY	PROF_YEARS
Dysart	35000.00	3
Hall	45000.00	7
Steinbacner	30000.00	1
Bailey	50000.00	15
Clements	40000.00	4

## 10.4 DISTINCT

### Example 10.4.1

I want a list of all states from which this year's students come from, listing only the home state without any identifying student information.

```
-- Example 10.4.1

--      select ST_STATE
--      from STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.4.1");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( ST_STATE,
        FROM => STUDENT ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT ("ST_STATE");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_ST_STATE, V_ST_STATE_INDEX);
        GOT_ONE := GOT_ONE + 1;
```

UNCLASSIFIED

```
SET_COL (1); -- ST_STATE
PUT (V_ST_STATE, V_ST_STATE_INDEX);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

**Output of Example 10.4.1**

```
ST_STATE
VA
MD
PA
NC
VA
DC
MD
PA
VA
NC
PA
SC
NY
VA
WV
DC
MD
NY
SC
VA
PA
DC
NC
MD
VA
```

We have a list containing several duplicate states. We want just a list of the home states with each state listed only once. To produce such a list we would use the `SELECT_DISTINCT` statement instead of the `SELEC` statement. This would list each distinctive state only once. If more than one column is selected when the `SELECT_DISTINCT` then each record listed will not duplicate any other record

## UNCLASSIFIED

listed. The format of the SELECT\_DISTINCT is:

```
SELECT_DISTINCT ( column_1 & column_2 & ... ,  
FROM => table_1 ) ;
```

### Example 10.4.2

List the states from which our students come, list each state only once.

-- Example 10.4.2

```
--      select distinct ST_STATE  
--      from STUDENT ;
```

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.4.2");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>  
SELECT_DISTINCT ( ST_STATE,  
FROM => STUDENT ) );
```

```
OPEN ( CURSOR );
```

```
begin
```

```
NEW_LINE;  
PUT ("ST_STATE");  
GOT_ONE := 0;
```

```
loop
```

```
FETCH ( CURSOR );  
INTO (V_ST_STATE, V_ST_STATE_INDEX);  
GOT_ONE := GOT_ONE + 1;
```

```
SET_COL (1); -- ST_STATE  
PUT (V_ST_STATE, V_ST_STATE_INDEX);
```

```
NEW_LINE;  
end loop;
```

```
exception
```

```
when NOT_FOUND_ERROR => if GOT_ONE = 0 then  
PUT_LINE ("EXCEPTION: Not Found Error");  
else  
null;  
end if;
```

```
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
```

```
when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");  
end;
```

```
CLOSE ( CURSOR );
```



UNCLASSIFIED

Output of Example 10.4.2

ST\_STATE  
DC  
MD  
NC  
NY  
PA  
SC  
VA  
WV

Example 10.4.3

For each state represented by our student body do we have students attending their first, second, third or fourth year? I don't need to know the number of students in each category, only which categories exist in our student body.

```
-- Example 10.4.3

--      select distinct ST_STATE, ST_YEAR
--      from STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.4.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELECT_DISTINCT ( ST_STATE & ST_YEAR,
        FROM => STUDENT ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_STATE  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (11); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;
```

## UNCLASSIFIED

```
exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.4.3

ST_STATE	ST_YEAR
DC	ONE
DC	TWO
DC	THREE
MD	ONE
MD	THREE
MD	FOUR
NC	ONE
NC	TWO
NC	THREE
NY	ONE
NY	FOUR
PA	TWO
PA	THREE
PA	FOUR
SC	TWO
VA	ONE
VA	THREE
VA	FOUR
WV	TWO

## 10.5 WHERE

You now know how to select all records or all distinctive records from a table. But frequently you will want to select only certain records based on specific criteria of one or more columns. To do this you add a WHERE statement after the FROM statement in a query. The format of the WHERE statement is:

```
SELEC ( columns,
  FROM => tables,
  WHERE => where_clause_comparison ) ;
```

## 10.6 WHERE With Comparison Operators = <> < <= > >=

The comparison operators available are:

equal to	EQ ( left, right )
not equal to	NE ( left, right )
less than	<
less than or equal to	<=
greater than	>
greater than or equal to	>=

Equal and not equal use the functions EQ and NE instead of = and <>. Ada does not allow the overloading of the = or the <> functions. The format of the WHERE statement with a comparison operator is:

```
SELEC ( columns,
  FROM => tables,
  WHERE => column/variable COMPARISON OPERATOR column/variable ) ;
```

And with the EQ or NE function is:

```
SELEC ( columns,
  FROM => tables,
  WHERE => EQ/NE ( column/variable, column/variable ) ) ;
```

### Example 10.6.1

Let's select all students from Virginia.

```
-- Example 10.6.1

--      select *
--      from STUDENT
--      where ST_STATE = 'VA' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.1");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
  FROM => STUDENT,
  WHERE => EQ ( ST_STATE, "VA" ) ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
```

# UNCLASSIFIED

```

GOT_ONE := 0;

loop
  FETCH ( CURSOR );
  INTO (V_ST_ID);
  INTO (V_ST_NAME, V_ST_NAME_INDEX);
  INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
  INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
  INTO (V_ST_STATE, V_ST_STATE_INDEX);
  INTO (V_ST_MAJOR);
  INTO (V_ST_YEAR);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- ST_ID
  PUT (V_ST_ID, 3);
  SET_COL (8); -- ST_NAME
  PUT (V_ST_NAME, V_ST_NAME_INDEX);
  SET_COL (22); -- ST_FIRST
  PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
  SET_COL (34); -- ST_ROOM
  PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
  SET_COL (43); -- ST_STATE
  PUT (V_ST_STATE, V_ST_STATE_INDEX);
  SET_COL (53); -- ST_MAJOR
  PUT (V_ST_MAJOR, 1);
  SET_COL (63); -- ST_YEAR
  PUT (V_ST_YEAR);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.6.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR
5	McNamara	Howard	A201	VA	5	ONE
9	Bearman	Rose	A301	VA	2	ONE
14	Sherman	Donald	B102	VA	3	THREE
20	McMurray	Eric	B204	VA	2	ONE

## UNCLASSIFIED

25 Chateaufneuf Chelsea C105 VA

1 THREE

## Example 10.6.2

Remember when comparisons are to character strings enclose the strings in quotes and pad with spaces. Select the student record for McGinn.

-- Example 10.6.2

```
--      select *
--      from STUDENT
--      where ST_NAME = 'McGinn      ' ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.6.2");

DECLAR ( CURSOR , CURSOR\_FOR =&gt;

SELEC ( '\*' ,

FROM =&gt; STUDENT,

WHERE =&gt; EQ ( ST\_NAME, "McGinn " ) ) ) ;

OPEN ( CURSOR );

begin

NEW\_LINE;

```
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
     "ST_MAJOR  ST_YEAR");
```

GOT\_ONE := 0;

loop

FETCH ( CURSOR );

INTO (V\_ST\_ID);

INTO (V\_ST\_NAME, V\_ST\_NAME\_INDEX);

INTO (V\_ST\_FIRST, V\_ST\_FIRST\_INDEX);

INTO (V\_ST\_ROOM, V\_ST\_ROOM\_INDEX);

INTO (V\_ST\_STATE, V\_ST\_STATE\_INDEX);

INTO (V\_ST\_MAJOR);

INTO (V\_ST\_YEAR);

GOT\_ONE := GOT\_ONE + 1;

SET\_COL (1); -- ST\_ID

PUT (V\_ST\_ID, 3);

SET\_COL (8); -- ST\_NAME

PUT (V\_ST\_NAME, V\_ST\_NAME\_INDEX);

SET\_COL (22); -- ST\_FIRST

PUT (V\_ST\_FIRST, V\_ST\_FIRST\_INDEX);

SET\_COL (34); -- ST\_ROOM

PUT (V\_ST\_ROOM, V\_ST\_ROOM\_INDEX);

SET\_COL (43); -- ST\_STATE

PUT (V\_ST\_STATE, V\_ST\_STATE\_INDEX);

# UNCLASSIFIED

```

SET_COL (53); -- ST_MAJOR
PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
PUT (V_ST_YEAR);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.6.2

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
2	McGinn	Gregory	A102	MD	1	THREE

## Example 10.6.3

List all professors who are teaching for the first year at our school.

-- Example 10.6.3

```

--      select *
--      from PROFESSOR
--      where PROF_YEARS = 1 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.3");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => PROFESSOR,
    WHERE => EQ ( PROF_YEARS, 1 ) ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
    "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

```

# UNCLASSIFIED

```

loop
  FETCH ( CURSOR );
  INTO ( V_PROF_ID );
  INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
  INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
  INTO ( V_PROF_DEPT );
  INTO ( V_PROF_YEARS );
  INTO ( V_PROF_SALARY );
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- PROF_ID
  PUT (V_PROF_ID, 2);
  SET_COL (10); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
  SET_COL (24); -- PROF_FIRST
  PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
  SET_COL (36); -- PROF_DEPT
  PUT (V_PROF_DEPT, 1);
  SET_COL (47); -- PROF_YEARS
  PUT (V_PROF_YEARS, 2);
  SET_COL (59); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.6.3

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
3	Steinbacner	Moris	2	1	30000.00

## Example 10.6.4

List all professors who are not teaching at the school for the first year.

```

-- Example 10.6.4

--      select *

```

UNCLASSIFIED

```
--      from PROFESSOR
--      where PROF_YEARS <> 1 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.4");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => PROFESSOR,
              WHERE => NE ( PROF_YEARS, 1 ) ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
            "PROF_YEARS  PROF_SALARY");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
```



# UNCLASSIFIED

```

when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

```

CLOSE ( CURSOR );

```

## Output of Example 10.6.4

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

## Example 10.6.5

Or we could list all professors who have taught at the school for more than one year. The same criteria stated differently.

```

-- Example 10.6.5

```

```

--      select *
--      from PROFESSOR
--      where PROF_YEARS > 1 ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.6.5");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => PROFESSOR,
              WHERE => PROF_YEARS > 1 ) );

```

```

OPEN ( CURSOR );

```

```

begin

```

```

NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME    PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

```

```

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

```

# UNCLASSIFIED

```

SET_COL (1); -- PROF_ID
  PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
  PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
  PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
  PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.6.5

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

## Example 10.6.6

List all professors who have taught at the school for at least four years.

```

-- Example 10.6.6

--      select *
--      from PROFESSOR
--      where PROF_YEARS >= 4 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.6");

DECLAR ( CURSOR , CURSOR_FOR =>

```

# UNCLASSIFIED

```

SELEC ( '*',
FROM => PROFESSOR,
WHERE => PROF_YEARS >= 4 ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
"PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
FETCH ( CURSOR );
INTO ( V_PROF_ID );
INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
INTO ( V_PROF_DEPT );
INTO ( V_PROF_YEARS );
INTO ( V_PROF_SALARY );
GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
PUT_LINE ("EXCEPTION: Not Found Error");
else
null;
end if;

when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

**Output of Example 10.6.6**

# UNCLASSIFIED

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
2	Hall	Elizabeth	4	7	45000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

## Example 10.6.7

List all professors teaching for under four years.

-- Example 10.6.7

```
--      select *
--      from PROFESSOR
--      where PROF_YEARS < 4 ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.6.7");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => PROFESSOR,
              WHERE => PROF_YEARS < 4 ) );
```

```
OPEN ( CURSOR );
```

begin

```
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;
```

loop

```
FETCH ( CURSOR );
INTO ( V_PROF_ID );
INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
INTO ( V_PROF_DEPT );
INTO ( V_PROF_YEARS );
INTO ( V_PROF_SALARY );
GOT_ONE := GOT_ONE + 1;
```

```
SET_COL (1); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
```

# UNCLASSIFIED

```

        PUT (V_PROF_YEARS, 2);
        SET_COL (59); -- PROF_SALARY
        PUT (V_PROF_SALARY, 5, 2, 0);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR     => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.6.7

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
3	Steinbacner	Moris	2	1	30000.00

## Example 10.6.8

Or we could phrase it as, list all professors who have been with the school no more than three years.

-- Example 10.6.8

```

--      select *
--      from PROFESSOR
--      where PROF_YEARS <= 3 ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.6.8");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
        FROM => PROFESSOR,
        WHERE => PROF_YEARS <= 3 ) );

```

```

OPEN ( CURSOR );

```

```

begin
    NEW_LINE;
    PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
    GOT_ONE := 0;

```

# UNCLASSIFIED

```

loop
  FETCH ( CURSOR );
  INTO ( V_PROF_ID );
  INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
  INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
  INTO ( V_PROF_DEPT );
  INTO ( V_PROF_YEARS );
  INTO ( V_PROF_SALARY );
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- PROF_ID
  PUT (V_PROF_ID, 2);
  SET_COL (10); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
  SET_COL (24); -- PROF_FIRST
  PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
  SET_COL (36); -- PROF_DEPT
  PUT (V_PROF_DEPT, 1);
  SET_COL (47); -- PROF_YEARS
  PUT (V_PROF_YEARS, 2);
  SET_COL (59); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.6.8

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
3	Steinbacner	Moris	2	1	30000.00

## Example 10.6.9

We can compare one column to another, for example, list all classes where the students grades for the second semester were lower than their grades for the first semester. This compares the specified columns from the same record with each other. It will not compare columns from different records.

# UNCLASSIFIED

```
-- Example 10.6.9

--      select *
--      from CLASS
--      where CLASS_SEM_2 < CLASS_SEM_1 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.9");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => CLASS,
              WHERE => CLASS_SEM_2 < CLASS_SEM_1 ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("CLASS_STUDENT CLASS_DEPT CLASS_COURSE CLASS_SEM_1 " &
        "CLASS_SEM_2 CLASS_GRADE");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (V_CLASS_DEPT);
    INTO (V_CLASS_COURSE);
    INTO (V_CLASS_SEM_1);
    INTO (V_CLASS_SEM_2);
    INTO (V_CLASS_GRADE);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (15); -- CLASS_DEPT
    PUT (V_CLASS_DEPT, 1);
    SET_COL (26); -- CLASS_COURSE
    PUT (V_CLASS_COURSE, 3);
    SET_COL (39); -- CLASS_SEM_1
    PUT (V_CLASS_SEM_1, 3, 2, 0);
    SET_COL (51); -- CLASS_SEM_2
    PUT (V_CLASS_SEM_2, 3, 2, 0);
    SET_COL (63); -- CLASS_GRADE
    PUT (V_CLASS_GRADE, 3, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
```

# UNCLASSIFIED

```

        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.6.9

CLASS_STUDENT	CLASS_DEPT	CLASS_COURSE	CLASS_SEM_1	CLASS_SEM_2	CLASS_GRADE
1	3	302	89.49	51.91	0.00
4	2	204	71.17	70.55	0.00
5	5	503	88.83	81.12	0.00
8	5	502	69.68	56.92	0.00
11	4	401	81.99	76.29	0.00
14	3	302	92.27	82.47	0.00
16	1	101	85.64	78.26	0.00
16	1	101	94.59	91.52	0.00
17	4	401	94.71	63.36	0.00
18	5	503	92.69	71.69	0.00
20	2	204	88.28	79.01	0.00
23	4	402	96.33	81.53	0.00
24	5	503	97.14	85.72	0.00

## 10.7 WHERE With AND & OR

You can create rather complex selection criteria by adding the use of ANDs and ORs and selecting the precedence of the operators with parenthesis.

### Example 10.7.1

Select all students from Virginia here for the first year.

```

-- Example 10.7.1

--      select *
--      from STUDENT
--      where ST_STATE = 'VA' and ST_YEAR = 1 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.7.1");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => STUDENT,
    WHERE => EQ ( ST_STATE, "VA" )
    AND      EQ ( ST_YEAR, ONE ) ) );

```



# UNCLASSIFIED

```

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
       "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;

  exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
      PUT_LINE ("EXCEPTION: Not Found Error");
    else
      null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
  end;

  CLOSE ( CURSOR );

```

Output of Example 10.7.1

# UNCLASSIFIED

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
5	McNamara	Howard	A201	VA	5	ONE
9	Bearman	Rose	A301	VA	2	ONE
20	McMurray	Eric	B204	VA	2	ONE

## Example 10.7.2

List all students from North or South Carolina.

```
-- Example 10.7.2

--      select *
--      from STUDENT
--      where ST_STATE = 'NC' or ST_STATE = 'SC' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.7.2");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => STUDENT,
    WHERE => EQ ( ST_STATE, "NC" )
      OR      EQ ( ST_STATE, "SC" ) ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
```

# UNCLASSIFIED

```

SET_COL (34); -- ST_ROOM
PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
SET_COL (43); -- ST_STATE
PUT (V_ST_STATE, V_ST_STATE_INDEX);
SET_COL (53); -- ST_MAJOR
PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
PUT (V_ST_YEAR);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.7.2

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
4	Waxler	Dennis	A104	NC	2	TWO
10	Thompson	Paul	A302	NC	1	THREE
12	Schmidt	John	A304	SC	5	TWO
19	Phung	Kim	B203	SC	2	TWO
23	O'Day	Hilda	C103	NC	4	ONE

Note how we have to list the column ST\_STATE twice and cannot simply request ST\_STATE = 'NC' or 'SC'. ANDs and ORs must link complete comparisons not just the comparison values.

## Example 10.7.3

List all students from North or South Carolina and in their second year.

-- Example 10.7.3

```

--      select *
--      from STUDENT
--      where ( ST_STATE = 'NC' or ST_STATE = 'SC' )
--            and ST_YEAR = 2 ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.7.3");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>

```

## UNCLASSIFIED

```

        SELEC ( '*',
        FROM => STUDENT,
        WHERE => ( EQ ( ST_STATE, "NC" )
        OR          EQ ( ST_STATE, "SC" ) )
        AND          EQ ( ST_YEAR, TWO ) ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT ("ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
        "ST_MAJOR  ST_YEAR");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_ST_ID);
        INTO (V_ST_NAME, V_ST_NAME_INDEX);
        INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
        INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
        INTO (V_ST_STATE, V_ST_STATE_INDEX);
        INTO (V_ST_MAJOR);
        INTO (V_ST_YEAR);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- ST_ID
        PUT (V_ST_ID, 3);
        SET_COL (8); -- ST_NAME
        PUT (V_ST_NAME, V_ST_NAME_INDEX);
        SET_COL (22); -- ST_FIRST
        PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
        SET_COL (34); -- ST_ROOM
        PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
        SET_COL (43); -- ST_STATE
        PUT (V_ST_STATE, V_ST_STATE_INDEX);
        SET_COL (53); -- ST_MAJOR
        PUT (V_ST_MAJOR, 1);
        SET_COL (63); -- ST_YEAR
        PUT (V_ST_YEAR);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
        else
            null;
        end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

# UNCLASSIFIED

```
CLOSE ( CURSOR );
```

## Output of Example 10.7.3

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
4	Waxler	Dennis	A104	NC	2	TWO
12	Schmidt	John	A304	SC	5	TWO
19	Phung	Kim	B203	SC	2	TWO

## Example 10.7.4

List all professors who have taught for four years or less and earn a salary of more than \$33,000.00.

```
-- Example 10.7.4
```

```
--      select *
--      from PROFESSOR
--      where PROF_YEARS <= 4
--      and PROF_SALARY > 33000.00 ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.7.4");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*' ,
        FROM   => PROFESSOR ,
        WHERE  => PROF_YEARS <= 4
        AND    PROF_SALARY > 33000.00 ) );
```

```
OPEN ( CURSOR );
```

```
begin
```

```
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;
```

```
loop
```

```
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;
```

```
    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
```

# UNCLASSIFIED

```

    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.7.4

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
5	Clements	Carol	1	4	40000.00

We can generate quite complicated queries by combining multiple ANDs and ORs.

## Example 10.7.5

List all students from Virginia in their first year, all students from North or South Carolina in their second year, all students from Maryland in their third year, all students in their fourth year and all students from the District of Columbia.

```

-- Example 10.7.5

--      select *
--      from STUDENT
--      where ( ST_STATE = 'VA' and ST_YEAR = 1 )
--            or ( ( ST_STATE = 'NC' or ST_STATE = 'SC' )
--                  and ST_YEAR = 2 )
--            or ( ST_STATE = 'MD' and ST_YEAR = 3 )
--            or ST_YEAR = 4
--            or ST_STATE = 'DC' ;

```

# UNCLASSIFIED

```

NEW_LINE;
PUT_LINE ("Output of Example 10.7.5");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*' ,
    FROM => STUDENT,
    WHERE => ( EQ ( ST_STATE, "VA" ) and EQ ( ST_YEAR, ONE ) )
      OR      ( ( ( EQ (ST_STATE, "NC" ) or EQ ( ST_STATE, "SC" ) )
        and EQ (ST_YEAR, TWO ) )
      OR      ( ( EQ ( ST_STATE, "MD" ) and EQ ( ST_YEAR, THREE ) ) )
      OR      ( ( EQ ( ST_YEAR, FOUR ) ) )
      OR      ( ( EQ ( ST_STATE, "DC" ) ) ) ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then

```

# UNCLASSIFIED

```

        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.7.5

ST_ID	ST_NAME	ST_FIRST	ST_F00M	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR
2	McGinn	Gregory	A102	MD	1	THREE
4	Waxler	Dennis	A104	NC	2	TWO
5	McNamara	Howard	A201	VA	5	ONE
6	Hess	Fay	A202	DC	3	THREE
8	Hagan	Carl	A204	PA	5	FOUR
9	Bearman	Rose	A301	VA	2	ONE
12	Schmidt	John	A304	SC	5	TWO
13	Gevarter	Susan	B101	NY	5	FOUR
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR
19	Phung	Kim	B203	SC	2	TWO
20	McMurray	Eric	B204	VA	2	ONE
21	O'Leary	Peggy	C101	PA	3	FOUR
22	Martin	Charoltte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE

## 10.8 WHERE With BETWEEN Operator

If you want to select information based on a range of values you could do it with a complex query.

### Example 10.8.1

Let's list all professors who's salaries fall into a range of from \$35,000 to \$45,000.

```

-- Example 10.8.1

--      select *
--      from PROFESSOR
--      where PROF_SALARY >= 35000.00
--      and PROF_SALARY <= 45000.00 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.8.1");

```



UNCLASSIFIED

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => PROFESSOR,
    WHERE => PROF_SALARY >= 35000.00
      AND      PROF_SALARY <= 45000.00 ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
    "PROF_YEARS  PROF_SALARY");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

# UNCLASSIFIED

## Output of Example 10.8.1

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
5	Clements	Carol	1	4	40000.00

The between comparison operator allows you to select values in a range in a less awkward fashion. The format for the between operator is:

```

SELEC ( columns,
FROM => tables,
WHERE => BETWEEN ( column, lo_limit AND hi_limit ) ) ;

```

## Example 10.8.2

List all professors who's salaries fall into a range of from \$35,000 to \$45,000, using the between operator.

-- Example 10.8.2

```

--      select *
--      from PROFESSOR
--      where PROF_SALARY
--      between 35000.00 and 45000.00 ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.8.2");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
SELEC ( '*' ,
FROM => PROFESSOR,
WHERE => BETWEEN ( PROF_SALARY, 35000.00 and 45000.00 ) ) );

```

```

OPEN ( CURSOR );

```

begin

```

NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
"PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

```

loop

```

FETCH ( CURSOR );
INTO ( V_PROF_ID );
INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
INTO ( V_PROF_DEPT );
INTO ( V_PROF_YEARS );
INTO ( V_PROF_SALARY );

```

## UNCLASSIFIED

```
GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.8.2

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
5	Clements	Carol	1	4	40000.00

## 10.9 WHERE With The IN Operator

There are times when we may want to select column information from a list of possible constants. We could do this using ORs.

### Example 10.9.1

For example select all students from Virginia, Maryland and the District of Columbia.

-- Example 10.9.1

```
--      select *
--      from STUDENT
```

# UNCLASSIFIED

```

--          where ST_STATE = 'VA'
--          or   ST_STATE = 'MD'
--          or   ST_STATE = 'DC' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.9.1");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM  => STUDENT,
        WHERE => EQ ( ST_STATE, "VA" )
        OR     EQ ( ST_STATE, "MD" )
        OR     EQ ( ST_STATE, "DC" ) ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception

```

# UNCLASSIFIED

```

when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.9.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR
2	McGinn	Gregory	A102	MD	1	THREE
5	McNamara	Howard	A201	VA	5	ONE
6	Hess	Fay	A202	DC	3	THREE
7	Guiffre	Jennifer	A203	MD	4	ONE
9	Bearman	Rose	A301	VA	2	ONE
14	Sherman	Donald	B102	VA	3	THREE
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR
20	McMurray	Eric	B204	VA	2	ONE
22	Martin	Charoltte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE
25	Chateauneuf	Chelsea	C105	VA	1	THREE

Or we could simplify this query by using the IS\_IN statement, which allows us to select a column if its contents are equal to one item in a group. The format for the IS\_IN statement is:

```

SELECT ( columns,
        FROM => tables,
        WHERE => IS_IN ( option_1 OR option_2 OR ... ) ) ;

```

## Example 10.9.2

So the above example could be shortened to:

```

-- Example 10.9.2

--      select *
--      from STUDENT
--      where ST_STATE in ( 'VA', 'MD', 'DC' ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.9.2");

DECLAR ( CURSOR , CURSOR_FOR =>

```

UNCLASSIFIED

```

        SELEC ( '*',
        FROM => STUDENT,
        WHERE => IS_IN ( ST_STATE, "VA" or "MD" or "DC" ) ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT ("ST_ID ST_NAME ST_FIRST ST_ROOM ST_STATE " &
        "ST_MAJOR ST_YEAR");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_ST_ID);
        INTO (V_ST_NAME, V_ST_NAME_INDEX);
        INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
        INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
        INTO (V_ST_STATE, V_ST_STATE_INDEX);
        INTO (V_ST_MAJOR);
        INTO (V_ST_YEAR);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- ST_ID
        PUT (V_ST_ID, 3);
        SET_COL (8); -- ST_NAME
        PUT (V_ST_NAME, V_ST_NAME_INDEX);
        SET_COL (22); -- ST_FIRST
        PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
        SET_COL (34); -- ST_ROOM
        PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
        SET_COL (43); -- ST_STATE
        PUT (V_ST_STATE, V_ST_STATE_INDEX);
        SET_COL (53); -- ST_MAJOR
        PUT (V_ST_MAJOR, 1);
        SET_COL (63); -- ST_YEAR
        PUT (V_ST_YEAR);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## UNCLASSIFIED

### Output of Example 10.9.2

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR
2	McGinn	Gregory	A102	MD	1	THREE
5	McNamara	Howard	A201	VA	5	ONE
6	Hess	Fay	A202	DC	3	THREE
7	Guiffre	Jennifer	A203	MD	4	ONE
9	Bearman	Rose	A301	VA	2	ONE
14	Sherman	Donald	B102	VA	3	THREE
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR
20	McMurray	Eric	B204	VA	2	ONE
22	Martin	Charoltte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE
25	Chateaufneuf	Chelsea	C105	VA	1	THREE

### 10.10 Wild Characters

Wild characters are special characters used for the purpose of comparison with character strings. A percent symbol % matches any sequence of zero or more characters. An underscore \_ matches any one character. For example A% would match any character string, regardless of its length, if it began with the character A. A\_CDE would match an character string, five characters long, where the first character was A and the third, fourth and fifth were CDE and the second was any character.

### 10.11 WHERE With LIKE Operator

When comparing column values to constants with comparison operators or the IS\_IN statement the constant must match the data in the column exactly. But there may be times when you only want to match parts of column data using the wild characters described above. The LIKE statement is used when you wish to use pattern matching. The format of the LIKE statement is:

```
SELEC ( columns,
FROM => tables,
WHERE => LIKE ( column, pattern_matching_string ) ) ;
```

#### Example 10.11.1

To search for all names beginning with S you would use the pattern matching string 'S%'.

```
DECLAR ( CURSOR , CURSOR_FOR =>
SELEC ( ST_NAME,
FROM => STUDENT,
WHERE => LIKE ( ST_NAME, "S%" ) ) ) ;
```

You would assume that the above query would be used in this case. However ST\_NAME is a

# UNCLASSIFIED

constrained array and the Ada compiler checks for a constraint error here and will issue the message " %ADAC-I-CONSTRAINTS, (1) CONSTRAINT\_ERROR will be raised here [LRM 10.3.1, ". So to keep the compiler happy we have to fill the constant to the full length of the array. We do that using percent signs for the same effect. One percent sign would work with an unconstrained array.

-- Example 10.11.1

```
--      select ST_NAME
--      from STUDENT
--      where ST_NAME like 'S%' ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.11.1");

DECLAR ( CURSOR , CURSOR\_FOR =>

SELEC ( ST\_NAME,

FROM => STUDENT,

WHERE => LIKE ( ST\_NAME, "S%%%%%%%%%" ) ) ) ;

OPEN ( CURSOR );

begin

NEW\_LINE;

PUT ("ST\_NAME");

GOT\_ONE := 0;

loop

FETCH ( CURSOR );

INTO (V\_ST\_NAME, V\_ST\_NAME\_INDEX);

GOT\_ONE := GOT\_ONE + 1;

SET\_COL (1); -- ST\_NAME

PUT (V\_ST\_NAME, V\_ST\_NAME\_INDEX);

NEW\_LINE;

end loop;

exception

when NOT\_FOUND\_ERROR => if GOT\_ONE = 0 then

PUT\_LINE ("EXCEPTION: Not Found Error");

else

null;

end if;

when NO\_UPDATE\_ERROR => PUT\_LINE ("EXCEPTION: No Update Error");

when UNIQUE\_ERROR => PUT\_LINE ("EXCEPTION: Unique Error");

end;

CLOSE ( CURSOR );

**Output of Example 10.11.1**

ST\_NAME



UNCLASSIFIED

Schmidt  
Sherman

Example 10.11.2

To search for all students in dorm building A you would use the pattern matching string 'A%'.

-- Example 10.11.2

```
--      select ST_NAME, ST_ROOM
--      from STUDENT
--      where ST_ROOM like 'A%' ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.11.2");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_NAME & ST_ROOM,
        FROM => STUDENT,
        WHERE => LIKE ( ST_ROOM, "A%" ) ) ) ;
```

```
OPEN ( CURSOR );
```

begin

```
NEW_LINE;
PUT ("ST_NAME           ST_ROOM");
GOT_ONE := 0;
```

loop

```
FETCH ( CURSOR );
INTO (V_ST_NAME, V_ST_NAME_INDEX);
INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
GOT_ONE := GOT_ONE + 1;
```

```
SET_COL (1); -- ST_NAME
PUT (V_ST_NAME, V_ST_NAME_INDEX);
SET_COL (22); -- ST_ROOM
PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
NEW_LINE;
end loop;
```

exception

```
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
```

```
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
```

```
when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
```

end;

# UNCLASSIFIED

```
CLOSE ( CURSOR );
```

## Output of Example 10.11.2

ST_NAME	ST_ROOM
Horrigan	A101
McGinn	A102
Lewis	A103
Waxler	A104
McNamara	A201
Hess	A202
Guiffre	A203
Hagan	A204
Bearman	A301
Thompson	A302
Bennett	A303
Schmidt	A304

## Example 10.11.3

Or the pattern matching string 'A\_\_\_'.

```
-- Example 10.11.3
```

```
--      select ST_NAME, ST_ROOM
--      from STUDENT
--      where ST_ROOM like 'A___' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.11.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_NAME & ST_ROOM,
        FROM => STUDENT,
        WHERE => LIKE ( ST_ROOM, "A___" ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ( "ST_NAME          ST_ROOM");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    GOT_ONE := GOT_ONE + 1;
```

UNCLASSIFIED

```
SET_COL (1); -- ST_NAME
PUT (V_ST_NAME, V_ST_NAME_INDEX);
SET_COL (22); -- ST_ROOM
PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

Output of Example 10.11.3

ST_NAME	ST_ROOM
Horrigan	A101
McGinn	A102
Lewis	A103
Waxler	A104
McNamara	A201
Hess	A202
Guiffre	A203
Hagan	A204
Bearman	A301
Thompson	A302
Bennett	A303
Schmidt	A304

Example 10.11.4

To search for all students in room 101 of any dorm building you could use the pattern matching string '\_101'.

```
-- Example 10.11.4

--      select ST_NAME, ST_ROOM
--      from STUDENT
--      where ST_ROOM like '_101' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.11.4");
```

## UNCLASSIFIED

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( ST_NAME & ST_ROOM,
    FROM => STUDENT,
    WHERE => LIKE ( ST_ROOM, "_101" ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ( "ST_NAME          ST_ROOM" );
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.11.4

ST_NAME	ST_ROOM
Horrigan	A101
Gevarter	B101
O'Leary	C101

## 10.12 WHERE With The NOT Operator

There may be times you wish to select all records except those matching certain criteria. The NOT operator would be used here. It can be used with the comparison operators (EQ, NE, <, <=, >, >=), with the BETWEEN operator, the IS\_IN operator and the LIKE operator. When using the NOT operator the remainder of the clause must be surrounded by parenthesis. The format of the NOT

UNCLASSIFIED

operator in the WHERE statement is the same as the other operators but with the word not in front of the statement.

```
SELECT ( columns,
        FROM => tables,
        WHERE => NOT ( where_clause_comparison ) ) ;
```

The format of the NOT clause in the WHERE statement with comparison operators is :

```
SELEC ( columns,
        FROM => tables,
        WHERE => NOT ( column/constant/variable COMPARISON_OPERATOR
                      column/constant/variable ) ); or:
SELEC ( columns,
        FROM => tables,
        WHERE => NOT ( EQ/NE ( column/constant/variable,
                              column/constant/variable ) ) ) ;
```

The format of the NOT operator in the WHERE statement with the BETWEEN operator is:

```
SELEC ( columns,
        FROM => tables,
        WHERE => NOT ( BETWEEN ( column, lo_limit AND hi_limit ) ) ) ;
```

The format of the NOT operator in the WHERE statement with the IS\_IN operator is:

```
SELEC ( columns,
        FROM => tables,
        WHERE => NOT ( IS_IN ( column, option_1 OR option_2 OR ... ) ) ) ;
```

The format of the NOT operator in the WHERE statement with the LIKE operator is:

```
SELEC ( columns,
        FROM => tables,
        WHERE => NOT ( LIKE ( column, pattern_matching_string ) ) ) ;
```

Here are several examples of the NOT operator.

**Example 10.12.1**

Select all students who are not from Virginia.

```
-- Example 10.12.1

--      select *
--      from STUDENT
--      where not (ST_STATE = 'VA' ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.12.1");
```

# UNCLASSIFIED

```

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*' ,
        FROM => STUDENT,
        WHERE => NOT ( EQ ( ST_STATE, "VA" ) ) ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT ( "ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
        "ST_MAJOR  ST_YEAR");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_ST_ID);
        INTO (V_ST_NAME, V_ST_NAME_INDEX);
        INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
        INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
        INTO (V_ST_STATE, V_ST_STATE_INDEX);
        INTO (V_ST_MAJOR);
        INTO (V_ST_YEAR);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- ST_ID
        PUT (V_ST_ID, 3);
        SET_COL (8); -- ST_NAME
        PUT (V_ST_NAME, V_ST_NAME_INDEX);
        SET_COL (22); -- ST_FIRST
        PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
        SET_COL (34); -- ST_ROOM
        PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
        SET_COL (43); -- ST_STATE
        PUT (V_ST_STATE, V_ST_STATE_INDEX);
        SET_COL (53); -- ST_MAJOR
        PUT (V_ST_MAJOR, 1);
        SET_COL (63); -- ST_YEAR
        PUT (V_ST_YEAR);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

# UNCLASSIFIED

CLOSE ( CURSOR );

## Output of Example 10.12.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
2	McGinn	Gregory	A102	MD	1	THREE
3	Lewis	Molly	A103	PA	4	TWO
4	Waxler	Dennis	A104	NC	2	TWO
6	Hess	Fay	A202	DC	3	THREE
7	Guiffre	Jennifer	A203	MD	4	ONE
8	Hagan	Carl	A204	PA	5	FOUR
10	Thompson	Paul	A302	NC	1	THREE
11	Bennett	Nellie	A303	PA	4	THREE
12	Schmidt	John	A304	SC	5	TWO
13	Gevarter	Susan	B101	NY	5	FOUR
15	Gorham	Milton	B103	WV	2	TWO
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR
18	Ratliff	Ann	B202	NY	5	ONE
19	Phung	Kim	B203	SC	2	TWO
21	O'Leary	Peggy	C101	PA	3	FOUR
22	Martin	Charoltte	C102	DC	1	TWO
23	O'Day	Hilda	C103	NC	4	ONE
24	Martin	Edward	C104	MD	5	THREE

## Example 10.12.2

List all professors except those who have taught for four years or less and earn a salary of more than \$33,000.00.

-- Example 10.12.2

```
--      select *
--      from PROFESSOR
--      where not ( PROF_YEARS <= 4
--      and PROF_SALARY > 33000.00 ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.12.2");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
  FROM => PROFESSOR,
  WHERE => NOT ( PROF_YEARS <= 4
  AND PROF_SALARY > 33000.00 ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
```

# UNCLASSIFIED

```

PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
          "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.12.2

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00

## Example 10.12.3



# UNCLASSIFIED

List all professors who's salaries fall outside a range of from \$35,000 to \$45,000.

-- Example 10.12.3

```
--      select *
--      from PROFESSOR
--      where PROF_SALARY
--      not between 35000.00 and 45000.00 ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.12.3");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM => PROFESSOR,
        WHERE => NOT ( BETWEEN ( PROF_SALARY, 35000.00 and 45000.00 ) ) ) ) ;
```

```
OPEN ( CURSOR );
```

begin

```
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;
```

loop

```
FETCH ( CURSOR );
INTO ( V_PROF_ID );
INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
INTO ( V_PROF_DEPT );
INTO ( V_PROF_YEARS );
INTO ( V_PROF_SALARY );
GOT_ONE := GOT_ONE + 1;
```

```
SET_COL (1); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
```

```
NEW_LINE;
end loop;
```

exception

# UNCLASSIFIED

```

when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.12.3

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00

## Example 10.12.4

Select all students from anywhere except Virginia, Maryland and the District of Columbia.

-- Example 10.12.4

```

--      select *
--      from STUDENT
--      where ST_STATE not in ( 'VA', 'MD', 'DC' ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.12.4");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => STUDENT,
    WHERE => NOT ( IS_IN ( ST_STATE, "VA" or "MD" or "DC" ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_ID  ST_NAME          ST_FIRST    ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);

```

UNCLASSIFIED

```

    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

Output of Example 10.12.4

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
3	Lewis	Molly	A103	PA	4	TWO
4	Waxler	Dennis	A104	NC	2	TWO
8	Hagan	Carl	A204	PA	5	FOUR
10	Thompson	Paul	A302	NC	1	THREE
11	Bennett	Nellie	A303	PA	4	THREE
12	Schmidt	John	A304	SC	5	TWO
13	Gevarter	Susan	B101	NY	5	FOUR
15	Gorham	Milton	B103	WV	2	TWO
18	Ratliff	Ann	B202	NY	5	ONE
19	Phung	Kim	B203	SC	2	TWO
21	O'Leary	Peggy	C101	PA	3	FOUR
23	O'Day	Hilda	C103	NC	4	ONE

# UNCLASSIFIED

## Example 10.12.5

Search for all students in all dorm buildings except building A.

-- Example 10.12.5

```
--      select ST_NAME, ST_ROOM
--      from STUDENT
--      where ST_ROOM not like 'A%' ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.12.5");

DECLAR ( CURSOR , CURSOR\_FOR =>

SELEC ( ST\_NAME & ST\_ROOM,

FROM => STUDENT,

WHERE => NOT ( LIKE ( ST\_ROOM, "A%" ) ) ) ;

OPEN ( CURSOR );

begin

NEW\_LINE;

PUT ("ST\_NAME ST\_ROOM");

GOT\_ONE := 0;

loop

FETCH ( CURSOR );

INTO (V\_ST\_NAME, V\_ST\_NAME\_INDEX);

INTO (V\_ST\_ROOM, V\_ST\_ROOM\_INDEX);

GOT\_ONE := GOT\_ONE + 1;

SET\_COL (1); -- ST\_NAME

PUT (V\_ST\_NAME, V\_ST\_NAME\_INDEX);

SET\_COL (22); -- ST\_ROOM

PUT (V\_ST\_ROOM, V\_ST\_ROOM\_INDEX);

NEW\_LINE;

end loop;

exception

when NOT\_FOUND\_ERROR => if GOT\_ONE = 0 then

PUT\_LINE ("EXCEPTION: Not Found Error");

else

null;

end if;

when NO\_UPDATE\_ERROR => PUT\_LINE ("EXCEPTION: No Update Error");

when UNIQUE\_ERROR => PUT\_LINE ("EXCEPTION: Unique Error");

end;

CLOSE ( CURSOR );

**Output of Example 10.12.5**

ST_NAME	ST_ROOM
Gevarter	B101
Sherman	B102
Gorham	B103
Williams	B104
Woodliff	B201
Ratliff	B202
Phung	B203
McMurray	B204
O'Leary	C101
Martin	C102
O'Day	C103
Martin	C104
Chateauneuf	C105

**10.13 The Arithmetic Expressions + - \* /**

You may wish to perform arithmetic calculations on the data in numeric columns for display purposes or for the purpose of comparison. You can use an arithmetic expression by connecting numeric columns and or numeric constants or variables with the arithmetic operators:

```
+ add
- subtract
* multiply
/ divide
```

You may use parenthesis to establish precedence of operations within an expression. Arithmetic expressions may be used wherever a column name is allowed. An arithmetic operation may be performed between one or more numeric fields and/or using one or more numeric constants or variables.

An arithmetic expression may be used in place of a column name in the list of columns to select. Why would you want to do this? Imagine you'd like to see what salaries would be if everyone got an 10% raise, but you don't really want to change the data. So you would select salary \* 1.10. The format of an arithmetic expression as an item in a select list is:

```
SELEC ( ( column_or_constant arithmetic_operator column_or_operator ) ... ,
FROM ...
```

**Example 10.13.1**

List the professors and their salaries if they were to receive a 10% raise.

```
-- Example 10.13.1

--      select PROF_NAME, PROF_SALARY * 1.10
--      from PROFESSOR ;

NEW_LINE;
PUT_LINE ("Output of Example 10.13.1");
```

# UNCLASSIFIED

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( PROF_NAME & ( PROF_SALARY * 1.10 ),
    FROM => PROFESSOR ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("PROF_NAME      PROF_SALARY * 1.10");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (15); -- PROF_SALARY * 1.10
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;

  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.13.1

PROF_NAME	PROF_SALARY * 1.10
Dysart	38500.00
Hall	49500.00
Steinbacner	33000.00
Bailey	55000.00
Clements	44000.00

## Example 10.13.2

List the average of the two semester grades for the classes without using the CLASS\_GRADE field, use only the semester grades.

# UNCLASSIFIED

-- Example 10.13.2

```
--      select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS ;
--

NEW_LINE;
PUT_LINE ("Output of Example 10.13.2");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
        FROM => CLASS) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("CLASS_STUDENT      CLASS_SEM_1 + CLASS_SEM_2 / 2");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (V_CLASS_SEM_1);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (19); -- CLASS_SEM_1 + CLASS_SEM_2 / 2
    PUT (V_CLASS_SEM_1, 3, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

**Output of Example 10.13.2**

CLASS_STUDENT	CLASS_SEM_1 + CLASS_SEM_2 / 2
1	70.70
1	83.22

# UNCLASSIFIED

2	69.57
3	95.20
4	70.86
5	84.97
6	80.43
6	100.00
7	100.00
7	100.00
7	100.00
7	100.00
8	63.30
9	72.67
10	96.64
11	79.14
12	79.42
13	73.75
14	87.37
15	92.75
16	81.95
16	93.06
16	89.14
16	84.63
16	93.66
16	86.29
17	79.04
18	82.19
19	88.63
20	83.64
21	72.65
22	72.78
22	87.36
22	86.39
23	88.93
24	91.43
25	86.37

NOTE: I have to use 2.00 in the select instead of 2 to avoid a comparison error with the application scanner "%ADASQL-E-SCAN, Operands not comparable". Also keep in mind that the fractional answers may vary slightly from the ones printed with the ad hoc queries, this is due to Ada rounding in the print routines vs rounding in the DBMS. We get the number back from the DBMS as a floating point number and have to then perform the conversion to ASCII for printing.

An arithmetic expression may be used in place of a column name as selection criteria in a where clause. For example in a table which included current salary and previous years salary you might want to select anyone who's salary is greater than 10% more than last years salary. The format of an arithmetic expression in a where clause is:

```
... column/constant/variable arithmetic_operator
      column/constant/variable ...
```



UNCLASSIFIED

Example 10.13.3

List the professors who have a salary which is greater than \$10,000 for each year of their employment.

NOTE: in order to compare two columns, constants or variables they must be of the same data types. In this example we wish to compare PROF\_SALARY of data type YEARLY\_INCOME to PROF\_YEARS of data type YEARS\_EMPLOYED, multiplied by 10,000. Ada will not permit this without type conversions. The Ada/SQL conversion function must be used to convert COLUMNS to similar data types. The format of the Ada/SQL conversion is:

```
CONVERT_TO.outer_package_name.column_data_type ( column_name )
```

Where outer\_package\_name is the name of the outer package of your type descriptor package where the data types are defined for the columns. You do not use the inner package, ADA\_SQL, here. Column\_data\_type is the data type to which you wish to convert the data in the column of column\_name to. For example to convert PROF\_YEARS to the same data type as PROF\_SALARY you would use the conversion:

```
CONVERT_TO.TYPES.YEARLY_INCOME ( PROF_YEARS )
```

If you forget to use the conversion function here you will get an application scanner error of "%ADASQL-E-SCAN, Operands not comparable".

-- Example 10.13.3

```
--      select PROF_NAME, PROF_SALARY
--      from PROFESSOR
--      where PROF_SALARY > PROF_YEARS * 10000.00 ;
```

```
NEW_LINE;
```

```
PUT_LINE ("Output of Example 10.13.3");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
```

```
  SELEC ( PROF_NAME & PROF_SALARY,
```

```
  FROM => PROFESSOR,
```

```
  WHERE => PROF_SALARY >
```

```
    ( CONVERT_TO.TYPES.YEARLY_INCOME ( PROF_YEARS ) * 10000.00 ) ) ) ;
```

```
OPEN ( CURSOR );
```

```
begin
```

```
  NEW_LINE;
```

```
  PUT_LINE ("PROF_NAME      PROF_SALARY");
```

```
  GOT_ONE := 0;
```

```
  loop
```

```
    FETCH ( CURSOR );
```

```
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
```

```
    INTO ( V_PROF_SALARY );
```

```
    GOT_ONE := GOT_ONE + 1;
```

# UNCLASSIFIED

```

SET_COL (1);  -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (15);  -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.13.3

PROF_NAME	PROF_SALARY
Dysart	35000.00
Steinbacner	30000.00

## Example 10.13.4

Now if we were to give our professors a 10% raise which ones would be making less than \$10000.00 for each year of employment. Remember to use conversions where necessary.

```

-- Example 10.13.4

--      select PROF_NAME, PROF_SALARY * 1.10
--      from PROFESSOR
--      where PROF_SALARY * 1.10 < PROF_YEARS * 10000.00 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.13.4");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( PROF_NAME & ( PROF_SALARY * 1.10 ),
    FROM => PROFESSOR,
    WHERE => PROF_SALARY * 1.10 <
        CONVERT_TO.TYPES.YEARLY_INCOME ( PROF_YEARS ) * 10000.00 ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;

```

## UNCLASSIFIED

```
PUT_LINE ("PROF_NAME      PROF_SALARY * 1.10");
GOT_ONE := 0;

loop
  FETCH ( CURSOR );
  INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
  INTO ( V_PROF_SALARY );
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
  SET_COL (15); -- PROF_SALARY * 1.10
  PUT (V_PROF_SALARY, 5, 2, 0);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.13.4

PROF_NAME	PROF_SALARY * 1.10
Hall	49500.00
Bailey	55000.00

## 10.14 The Aggregate Functions COUNT, MIN, MAX, SUM, AVG

The aggregate functions are applied to all selected records in a table and provide information using data from each record. These functions return summary information about groups of records in the table. For example we can list a count of the student body, the minimum or maximum salary paid to a professor, the totally expenditure in salaries, the average class grades for a student. These functions may be applied to column names in the list of columns to select. The format of the aggregate functions is:

```
aggregate_function ( column_name )
```

The aggregate functions available are:

- COUNT - the number of values in the column chosen
- MIN - the minimum value in the column chosen

## UNCLASSIFIED

MAX - the maximum value in the column chosen  
SUM - the total of the values in the column chosen  
AVG - the average of the values in the column chosen

Only one row will be listed as output and any column selected to be listed must apply to all records to be selected as part of the group selected. For example if you were to list a count of the student body it would be inappropriate to request that student name be listed. However if you were to list minimum salary for professors you could request the name also. But what would happen if more than one professor earned the same salary which was the lowest salary? Your data would be erroneous since only one name would be listed. So please, to avoid confusion, list only data which applies to all records that might be used to create the requested information.

The package text\_io cannot be "use"d in a unit which also uses the COUNT aggregate in a select clause. If text\_io is used you will get an error due to conflict of names. To get around this problem we renamed the routines in text\_io which we intended to use and put them in the program "conversions". A select of COUNT returns a value of the data type DATABASE.INT. All other aggregate functions return values of the data type of the column on which the aggregate function is preformed.

When calculating an aggregate function first all records are selected based on the criteria in the where clause. Then the function is applied to the aggregate fields and one row of information is displayed. Aggregate functions may be used only in a select clause (or a having clause which we will cover later on) and may never be used as selection criteria in a where clause. Remember that these functions will display one and only one row of information which is an aggregate of all the records selected based on criteria in the where clause.

### Example 10.14.1

List the count of the student body, which would simple be a count of the records in the STUDENT table, since each student gets one and only one record. We do not need the DECLAR clause here since we are only selecting one record.

```
-- Example 10.14.1

--      select count (*)
--      from STUDENT ;
--

begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.14.1");

  SELEC ( COUNT ('*'),
    FROM => STUDENT );
    INTO (COUNT_RESULT);

  NEW_LINE;
  PUT ("COUNT");
  SET_COL (1); -- COUNT
  PUT (COUNT_RESULT, 3);
  NEW_LINE;
```

UNCLASSIFIED

```
exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

**Output of Example 10.14.1**

```
COUNT
  25
```

**Example 10.14.2**

Now count the number of students in dorm building A.

-- Example 10.14.2

```
--      select count (*)
--      from STUDENT
--      where ST_ROOM like 'A%';
```

```
begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.14.2");

  SELEC ( COUNT ('*'),
    FROM => STUDENT,
    WHERE => LIKE ( ST_ROOM, "A%" ) );
    INTO (COUNT_RESULT);

  NEW_LINE;
  PUT ("COUNT");
  SET_COL (1); -- COUNT
  PUT (COUNT_RESULT, 3);
  NEW_LINE;
```

```
exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

**Output of Example 10.14.2**

```
COUNT
  12
```

**Example 10.14.3**

UNCLASSIFIED

Count the number of students from the Washington DC area, include Virginia and Maryland.

-- Example 10.14.3

```
--      select count (*)
--      from STUDENT
--      where ST_STATE in ( 'DC', 'VA', 'MD' ) ;

begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.14.3");

  SELEC ( COUNT ( '*' ),
    FROM => STUDENT,
    WHERE => IS_IN (ST_STATE, "DC" or "VA" or "MD" ) );
    INTO (COUNT_RESULT);

  NEW_LINE;
  PUT ("COUNT");
  SET_COL (1); -- COUNT
  PUT (COUNT_RESULT, 3);
  NEW_LINE;

exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

Output of Example 10.14.3

```
COUNT
  13
```

Example 10.14.4

Now count the number of students from Virginia and list the state being counted. I'm assuming it's ok to list the state here since all states being selected contain the same information in the column being listed.

```
SELEC ( ST_STATE & COUNT ( '*' ),
  FROM => STUDENT,
  WHERE => EQ ( ST_STATE, "VA" ) );
  INTO (V_ST_STATE, V_ST_STATE_INDEX);
  INTO (COUNT_RESULT);
```

If you run this example you will get an exception at execution time of "%ADA-F-EXCEPTION, Exception UNHANDLED\_RDBMS\_ERROR". The DBMS is not going to allow us to list a column which could result in erroneous results. The DBMS is disallowing this query since if we were counting all

## UNCLASSIFIED

records, the values for state would not be the same in all records. It is not taking in consideration the fact that we have narrowed the selection to only states of VA. Some DBMSs may permit such a query.

If you're building a large program to run all of these queries at the end do not include those which result in exceptions. Some DBMSs will back out all changes made to databases by a program which ends in an exception.

### Example 10.14.5

Let's try the query without the offending column selection.

```
-- Example 10.14.5

--      select count (*)
--      from STUDENT
--      where ST_STATE = 'VA' ;

begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.14.5");

  SELEC ( COUNT ( '*' ),
  FROM => STUDENT,
  WHERE => EQ ( ST_STATE, "VA" ) );
  INTO (COUNT_RESULT);

  NEW_LINE;
  PUT ("COUNT");
  SET_COL (1); -- COUNT
  PUT (COUNT_RESULT, 3);
  NEW_LINE;

exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

### Output of Example 10.14.5

```
COUNT
  6
```

### Example 10.14.6

The CLASS table lists each class taken by each student. I want a count of the unique classes being taken by all the students. I can use the COUNT\_DISTINCT function instead of the COUNT function in conjunction with an aggregate function to obtain the answer.

UNCLASSIFIED

```
SELEC ( COUNT_DISTINCT ( CLASS_COURSE ),
        FROM => CLASS ) ;
        INTO (COUNT_RESULT);
```

If you try to scan such a query you will get the scanner error "%ADASQL-E-SCAN, Identifier has no valid meaning in this context" because COUNT\_DISTINCT was not included in Level 1 of Ada/SQL. It will be included in future versions of Ada/SQL but for now it is an option not available to you.

**Example 10.14.7**

Now let's list the minimum and maximum salary paid to the professors.

-- Example 10.14.7

```
--      select min (PROF_SALARY), max (PROF_SALARY)
--      from PROFESSOR ;
```

begin

```
NEW_LINE;
PUT_LINE ("Output of Example 10.14.7");

SELEC ( min (PROF_SALARY) & max (PROF_SALARY),
        FROM => PROFESSOR );
        INTO ( MIN_SALARY );
        INTO ( MAX_SALARY );
```

```
NEW_LINE;
PUT_LINE ("MINIMUM SALARY      MAXIMUM SALARY");
SET_COL (1); -- MIN SALARY
PUT (MIN_SALARY, 7, 2, 0);
SET_COL (20); -- MAX SALARY
PUT (MAX_SALARY, 7, 2, 0);
NEW_LINE;
```

exception

```
when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

**Output of Example 10.14.7**

MINIMUM SALARY	MAXIMUM SALARY
30000.00	50000.00

**Example 10.14.8**

Add all professor's salaries together to list the total salary expenditure.



# UNCLASSIFIED

-- Example 10.14.8

```
--      select sum (PROF_SALARY)
--      from PROFESSOR ;
```

begin

NEW\_LINE;

PUT\_LINE ("Output of Example 10.14.8");

```
SELEC ( CONVERT_TO.TYPES.TOTAL_INCOME (sum (PROF_SALARY)),
FROM => PROFESSOR );
INTO ( SUM_SALARY );
```

NEW\_LINE;

PUT\_LINE ("SALARY");

SET\_COL (1); -- SUM SALARY

PUT (SUM\_SALARY, 7, 2, 0);

NEW\_LINE;

exception

when NOT\_FOUND\_ERROR => PUT\_LINE ("EXCEPTION: Not Found Error");

when NO\_UPDATE\_ERROR => PUT\_LINE ("EXCEPTION: No Update Error");

when UNIQUE\_ERROR => PUT\_LINE ("EXCEPTION: Unique Error");

end;

## Output of Example 10.14.8

```
SALARY
200000.00
```

Note the CONVERT\_TO applied to the sum of PROF\_SALARY. The column data would be returned as the data type of PROF\_SALARY, YEARLY\_INCOME. The upper constraint of YEARLY\_INCOME is 99,999. The total of all salaries will exceed this value causing a constraint error. Therefore we have defined the data type TOTAL\_INCOME which holds more digits and we must convert the column output to that data type.

## Example 10.14.9

Now list the average first and second semester grades for all classes taken by student Alvin Williams who is student number 016.

-- Example 10.14.9

```
--      select avg (CLASS_SEM_1), avg (CLASS_SEM_2)
--      from CLASS
--      where CLASS_STUDENT = 016 ;
```

# UNCLASSIFIED

```

begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.14.9");

  SELEC ( avg (CLASS_SEM_1) & avg (CLASS_SEM_2),
    FROM => CLASS,
    WHERE => EQ (CLASS_STUDENT, 016) );
    INTO (AVG_SEM_1);
    INTO (AVG_SEM_2);

  NEW_LINE;
  PUT_LINE ("AVERAGE CLASS_SEM_1      AVERAGE CLASS_SEM_2");
  SET_COL (1); -- AVG_SEM_1
  PUT (AVG_SEM_1, 3, 2, 0);
  SET_COL (25); -- AVG_SEM_2
  PUT (AVG_SEM_2, 3, 2, 0);
  NEW_LINE;

exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

## Output of Example 10.14.9

AVERAGE CLASS_SEM_1	AVERAGE CLASS_SEM_2
85.43	90.82

## Example 10.14.10

List the number of professors at our school, our total salary expenditure, the average salary per professor and the minimum and maximum salaries paid.

-- Example 10.14.10

```

--      select count (*), sum (PROF_SALARY), avg (PROF_SALARY),
--          min (PROF_SALARY), max (PROF_SALARY)
--      from PROFESSOR ;

```

```

begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.14.10");

  SELEC ( COUNT ('*') & CONVERT_TO.TYPES.TOTAL_INCOME (sum (PROF_SALARY)) &
    avg (PROF_SALARY) & min (PROF_SALARY) & max (PROF_SALARY),
    FROM => PROFESSOR );
    INTO ( COUNT_RESULT );
    INTO ( SUM_SALARY );

```

UNCLASSIFIED

```
INTO ( AVG_SALARY );
INTO ( MIN_SALARY );
INTO ( MAX_SALARY );

NEW_LINE;
PUT_LINE ("COUNT SALARY SUM AVERAGE MINIMUM MAXIMUM");
SET_COL (1); -- COUNT
PUT (COUNT_RESULT, 3);
SET_COL (15); -- SUM SALARY
PUT (SUM_SALARY, 9, 2, 0);
SET_COL (28); -- AVG SALARY
PUT (AVG_SALARY, 7, 2, 0);
SET_COL (39); -- MIN SALARY
PUT (MIN_SALARY, 7, 2, 0);
SET_COL (50); -- MAX SALARY
PUT (MAX_SALARY, 7, 2, 0);
NEW_LINE;

exception
when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

**Output of Example 10.14.10**

COUNT	SALARY	SUM	AVERAGE	MINIMUM	MAXIMUM
5	200000.00	40000.00	30000.00	50000.00	

Again note the conversion necessary in the sum of professor's salaries.

**10.15 ORDER BY**

So far we've just asked to see data and not paid any attention to the order in which the output was displayed. You will usually want to order your lists in some way. You can do this using the `ORDER_BY` clause. The format of a query with an `ORDER_BY` clause is:

```
SELEC ( columns,
FROM => tables,
WHERE => where_clause ),
ORDER_BY => column_name & column_name & ... ;
```

Multiple columns may be listed, each subsequent column will be sorted as a subset of the previous column. Each column may specify if the sort sequence is to be ascending, `ASC`, or descending, `DESC`. The format to use `ASC` or `DESC` is:

```
ASC ( column_name )
DESC ( column_name )
```

# UNCLASSIFIED

Ascending, from smallest such as A or 1 to the largest, Z or 9, is the default.

## Example 10.15.1

List all students in order of last name.

```
-- Example 10.15.1

--      select *
--      from STUDENT
--      order by ST_NAME ;

NEW_LINE;
PUT_LINE ("Output of Example 10.15.1");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => STUDENT ),
        ORDER_BY => ST_NAME );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
     "ST_MAJOR  ST_YEAR");
GOT_ONE := 0;

loop
  FETCH ( CURSOR );
  INTO (V_ST_ID);
  INTO (V_ST_NAME, V_ST_NAME_INDEX);
  INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
  INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
  INTO (V_ST_STATE, V_ST_STATE_INDEX);
  INTO (V_ST_MAJOR);
  INTO (V_ST_YEAR);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- ST_ID
  PUT (V_ST_ID, 3);
  SET_COL (8); -- ST_NAME
  PUT (V_ST_NAME, V_ST_NAME_INDEX);
  SET_COL (22); -- ST_FIRST
  PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
  SET_COL (34); -- ST_ROOM
  PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
  SET_COL (43); -- ST_STATE
  PUT (V_ST_STATE, V_ST_STATE_INDEX);
  SET_COL (53); -- ST_MAJOR
```

# UNCLASSIFIED

```

    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;

when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.15.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
9	Bearman	Rose	A301	VA	2	ONE
11	Bennett	Nellie	A303	PA	4	THREE
25	Chateaufneuf	Chelsea	C105	VA	1	THREE
13	Gevarter	Susan	B101	NY	5	FOUR
15	Gorham	Milton	B103	WV	2	TWO
7	Guiffre	Jennifer	A203	MD	4	ONE
8	Hagan	Carl	A204	PA	5	FOUR
6	Hess	Fay	A202	DC	3	THREE
1	Horriggan	William	A101	VA	3	FOUR
3	Lewis	Molly	A103	PA	4	TWO
22	Martin	Charoltte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE
2	McGinn	Gregory	A102	MD	1	THREE
20	McMurray	Eric	B204	VA	2	ONE
5	McNamara	Howard	A201	VA	5	ONE
23	O'Day	Hilda	C103	NC	4	ONE
21	O'Leary	Peggy	C101	PA	3	FOUR
19	Phung	Kim	B203	SC	2	TWO
18	Ratliff	Ann	B202	NY	5	ONE
12	Schmidt	John	A304	SC	5	TWO
14	Sherman	Donald	B102	VA	3	THREE
10	Thompson	Paul	A302	NC	1	THREE
4	Waxler	Dennis	A104	NC	2	TWO
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR

## Example 10.15.2

# UNCLASSIFIED

List all professors and their salaries with the largest salary first.

-- Example 10.15.2

```
--      select PROF_NAME, PROF_SALARY
--      from PROFESSOR
--      order by PROF_SALARY desc ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.15.2");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_NAME & PROF_SALARY,
        FROM => PROFESSOR ),
        ORDER_BY => DESC (PROF_SALARY) ) ;
```

```
OPEN ( CURSOR );
```

begin

```
NEW_LINE;
PUT_LINE ("PROF_NAME      PROF_SALARY");
GOT_ONE := 0;
```

loop

```
  FETCH ( CURSOR );
  INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
  INTO ( V_PROF_SALARY );
  GOT_ONE := GOT_ONE + 1;
```

```
  SET_COL (1); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
  SET_COL (15); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
  NEW_LINE;
end loop;
```

exception

```
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                          PUT_LINE ("EXCEPTION: Not Found Error");
                          else
                            null;
                          end if;
```

```
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
```

end;

```
CLOSE ( CURSOR );
```

**Output of Example 10.15.2**

# UNCLASSIFIED

PROF_NAME	PROF_SALARY
Bailey	50000.00
Hall	45000.00
Clements	40000.00
Dysart	35000.00
Steinbacner	30000.00

## Example 10.15.3

List all students by the number of years they have studied and the major they are studying. List the students with the most number of years first.

```
-- Example 10.15.3

--      select ST_NAME, ST_YEAR, ST_MAJOR
--      from STUDENT
--      order by ST_YEAR desc, ST_MAJOR ;

NEW_LINE;
PUT_LINE ("Output of Example 10.15.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_NAME & ST_YEAR & ST_MAJOR,
        FROM => STUDENT ),
        ORDER_BY => DESC (ST_YEAR) & ST_MAJOR );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_NAME      ST_YEAR  ST_MAJOR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_YEAR);
    INTO (V_ST_MAJOR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (15); -- ST_YEAR
    PUT (V_ST_YEAR);
    SET_COL (24); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
```

# UNCLASSIFIED

```

        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.15.3

ST_NAME	ST_YEAR	ST_MAJOR
Horrigan	FOUR	3
O'Leary	FOUR	3
Woodliff	FOUR	4
Hagan	FOUR	5
Gevarter	FOUR	5
McGinn	THREE	1
Chateauneuf	THREE	1
Thompson	THREE	1
Hess	THREE	3
Sherman	THREE	3
Bennett	THREE	4
Martin	THREE	5
Martin	TWO	1
Waxler	TWO	2
Gorham	TWO	2
Phung	TWO	2
Lewis	TWO	4
Schmidt	TWO	5
Williams	ONE	1
Bearman	ONE	2
McMurray	ONE	2
Guiffre	ONE	4
O'Day	ONE	4
McNamara	ONE	5
Ratliff	ONE	5

## Example 10.15.4

When sorting in ascending order I have omitted the ASC specification since it is the default. To include it in the above example we'd use:

```

-- Example 10.15.4

--      select ST_NAME, ST_YEAR, ST_MAJOR
--      from STUDENT
--      order by ST_YEAR desc, ST_MAJOR asc ;

```



# UNCLASSIFIED

```

NEW_LINE;
PUT_LINE ("Output of Example 10.15.4");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_NAME & ST_YEAR & ST_MAJOR,
                FROM => STUDENT ),
        ORDER_BY => DESC (ST_YEAR) & ASC (ST_MAJOR) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_NAME      ST_YEAR  ST_MAJOR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_YEAR);
    INTO (V_ST_MAJOR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (15); -- ST_YEAR
    PUT (V_ST_YEAR);
    SET_COL (24); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCFPTION: Nct Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.15.4

ST_NAME	ST_YEAR	ST_MAJOR
Horrigan	FOUR	3
O'Leary	FOUR	3
Woodliff	FOUR	4
Hagan	FOUR	5

# UNCLASSIFIED

Gevarter	FOUR	5
McGinn	THREE	1
Chateauneuf	THREE	1
Thompson	THREE	1
Hess	THREE	3
Sherman	THREE	3
Bennett	THREE	4
Martin	THREE	5
Martin	TWO	1
Waxler	TWO	2
Gorham	TWO	2
Phung	TWO	2
Lewis	TWO	4
Schmidt	TWO	5
Williams	ONE	1
Bearman	ONE	2
McMurray	ONE	2
Guiffre	ONE	4
O'Day	ONE	4
McNamara	ONE	5
Ratliff	ONE	5

## Example 10.15.5

For each class list the department, course, average for both semesters, and the student's id. List only those class where the average of the grade for the first semester and second semester is at least 90% and the grade of the second semester is at least 5% better than the grade for the first semester or where the student got a grade of 100% in either the first or second semester. Sort the list in the order of the highest second semester grade, highest first semester grade the department and course.

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_DEPT & CLASS_COURSE &
    ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) & CLASS_STUDENT,
  FROM => CLASS,
    WHERE => ( ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 >= 90.00
      AND CLASS_SEM_2 - CLASS_SEM_1 >= 5.00 )
      OR EQ (CLASS_SEM_1, 100.00)
      OR EQ (CLASS_SEM_2, 100.00) ) ),
  ORDER_BY => DESC (CLASS_SEM_2) & DESC (CLASS_SEM_1) &
    ASC (CLASS_DEPT) & ASC (CLASS_COURSE) ) ;

```

This is the the obvious query that comes to mind. However the application scanner will find the error "%ADASQL-E-SCAN, Column is not among those selected". I do not want to print out class\_sem\_1 and class\_sem\_2 but I must select them so they may be included in the order\_by, but note that I do not print. Any columns listed in the order\_by must be contained in the selec.

## -- Example 10.15.5

```

--      select CLASS_DEPT, CLASS_COURSE, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2,
--      CLASS_STUDENT
--      from CLASS

```

UNCLASSIFIED

```
--      where ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 >= 90.00
--            and CLASS_SEM_2 - CLASS_SEM_1 >= 5.00 )
--            or CLASS_SEM_1 = 100.00
--            or CLASS_SEM_2 = 100.00
--      order by CLASS_SEM_2 desc, CLASS_SEM_1 desc, CLASS_DEPT asc,
--            CLASS_COURSE asc ;

NEW_LINE;
PUT_LINE ("Output of Example 10.15.5");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_DEPT & CLASS_COURSE &
    ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) & CLASS_STUDENT &
    CLASS_SEM_1 & CLASS_SEM_2,
  FROM => CLASS,
  WHERE => ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 >= 90.00
    AND CLASS_SEM_2 - CLASS_SEM_1 >= 5.00 )
    OR      EQ (CLASS_SEM_1, 100.00)
    OR      EQ (CLASS_SEM_2, 100.00) ) ,
  ORDER_BY => DESC (CLASS_SEM_2) & DESC (CLASS_SEM_1) & ASC (CLASS_DEPT) &
    ASC (CLASS_COURSE) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_DEPT CLASS_COURSE CLASS_SEM_1 + CLASS_SEM_2 / 2 " &
    "CLASS_STUDENT");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_DEPT);
    INTO (V_CLASS_COURSE);
    INTO (AVG_SEM_1);
    INTO (V_CLASS_STUDENT);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_DEPT
    PUT (V_CLASS_DEPT, 1);
    SET_COL (13); -- CLASS_COURSE
    PUT (V_CLASS_COURSE, 3);
    SET_COL (28); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    SET_COL (60); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
```

## UNCLASSIFIED

```
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.15.5

CLASS_DEPT	CLASS_COURSE	CLASS_SEM_1 + CLASS_SEM_2 / 2	CLASS_STUDENT
4	401	100.00	7
4	402	100.00	6
4	402	100.00	7
4	403	100.00	7
5	503	100.00	7
1	102	96.64	10
4	403	93.66	16
2	202	92.75	15

Let me point out here that I have introduced the clauses, select, from, where, and order by in the order in which they must appear if they are to be in a query. An order by clause may never be followed by a from clause etc., it is always the last clause in a query.

## 10.16 GROUP BY

In one of the previous examples we wanted to know the average grades for all classes taken by one student. To get a list of the average grades for all students we would have to write a separate query for each student. How cumbersome. But there is a better way. We could use the GROUP\_BY clause which is used in conjunction with an aggregate function to perform a computation on common groups of records. In the past we used aggregate functions which treated all selected records as one group. By using the GROUP\_BY you may split your selected records into several groups and perform aggregate functions on each of those groups. The selected records are sorted and group breaks are made for each column in the GROUP\_BY clause. For each record listed the columns in the GROUP\_BY clause will be unique and the aggregate totals will be listed. The GROUP\_BY is always used in conjunction with an aggregate function. It has no meaning in a query without an aggregate function. The format of the GROUP\_BY clause in a query statement is:

```
SELECT ( columns_with aggregates,
        FROM => tables,
        WHERE => where_conditions,
        GROUP_BY => column & column & ... ) ;
```

### Example 10.16.1

For example to list the average first and second semester grades for all classes taken by all students we

# UNCLASSIFIED

would use the following statements.

```
-- Example 10.16.1

--      select CLASS_STUDENT, avg (CLASS_SEM_1), avg (CLASS_SEM_2)
--      from CLASS
--      group by CLASS_STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.16.1");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( CLASS_STUDENT & avg (CLASS_SEM_1) & avg (CLASS_SEM_2),
        FROM => CLASS,
        GROUP_BY => CLASS_STUDENT ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("CLASS_STUDENT      AVG_SEM_1      AVG_SEM_2");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (AVG_SEM_1);
    INTO (AVG_SEM_2);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (18); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    SET_COL (30); -- AVG_SEM_2
    PUT (AVG_SEM_2, 3, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

# UNCLASSIFIED

## Output of Example 10.16.1

CLASS_STUDENT	AVG_SEM_1	AVG_SEM_2
1	83.55	70.38
2	54.38	84.77
3	92.92	97.48
4	71.17	70.55
5	88.83	81.12
6	83.13	97.30
7	100.00	100.00
8	69.68	56.92
9	55.53	89.81
10	93.72	99.55
11	81.99	76.29
12	75.81	83.03
13	67.36	80.15
14	92.27	82.47
15	89.75	95.74
16	85.43	90.82
17	94.71	63.36
18	92.69	71.69
19	81.31	95.95
20	88.28	79.01
21	71.16	74.14
22	71.74	92.62
23	96.33	81.53
24	97.14	85.72
25	83.58	89.16

## Example 10.16.2

If we wanted the same listing but only for department 3 we would use the where clause in the following statements.

```
-- Example 10.16.2

--      select CLASS_STUDENT, avg (CLASS_SEM_1), avg (CLASS_SEM_2)
--      from CLASS
--      where CLASS_DEPT = 3
--      group by CLASS_STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.16.2");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & avg (CLASS_SEM_1) & avg (CLASS_SEM_2),
    FROM => CLASS,
    WHERE => EQ (CLASS_DEPT, 3),
    GROUP_BY => CLASS_STUDENT ) );
```

# UNCLASSIFIED

```

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_STUDENT      AVG_SEM_1      AVG_SEM_2");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (AVG_SEM_1);
    INTO (AVG_SEM_2);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (18); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    SET_COL (31); -- AVG_SEM_2
    PUT (AVG_SEM_2, 3, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.16.2

CLASS_STUDENT	AVG_SEM_1	AVG_SEM_2
1	83.55	70.38
6	66.26	94.60
14	92.27	82.47
16	82.14	87.11
21	71.16	74.14

## Example 10.16.3

Or if we wanted the listing of the student's grades but also broken down by department we would the following statements.

```
-- Example 10.16.3
```

UNCLASSIFIED

```
--      select CLASS_STUDENT, CLASS_DEPT, avg (CLASS_SEM_1), avg (CLASS_SEM_2)
--      from CLASS
--      group by CLASS_DEPT, CLASS_STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.16.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( CLASS_STUDENT & CLASS_DEPT & avg (CLASS_SEM_1) & avg (CLASS_SEM_2),
        FROM => CLASS.
        GROUP_BY => CLASS_DEPT & CLASS_STUDENT ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("CLASS_STUDENT   CLASS_DEPT   AVG_SEM_1   AVG_SEM_2");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (V_CLASS_DEPT);
    INTO (AVG_SEM_1);
    INTO (AVG_SEM_2);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (17); -- CLASS_DEPT
    PUT (V_CLASS_DEPT, 1);
    SET_COL (30); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    SET_COL (43); -- AVG_SEM_2
    PUT (AVG_SEM_2, 3, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```



# UNCLASSIFIED

## Output of Example 10.16.3

CLASS_STUDENT	CLASS_DEPT	AVG_SEM_1	AVG_SEM_2
2	1	54.38	84.77
10	1	93.72	99.55
16	1	90.11	84.89
22	1	58.97	86.58
25	1	83.58	89.16
4	2	71.17	70.55
9	2	55.53	89.81
15	2	89.75	95.74
16	2	83.40	94.88
19	2	81.31	95.95
20	2	88.28	79.01
22	2	81.75	92.97
1	3	83.55	70.38
6	3	66.26	94.60
14	3	92.27	82.47
16	3	82.14	87.11
21	3	71.16	74.14
3	4	92.92	97.48
6	4	100.00	100.00
7	4	100.00	100.00
11	4	81.99	76.29
16	4	89.92	97.40
17	4	94.71	63.36
23	4	96.33	81.53
5	5	88.83	81.12
7	5	100.00	100.00
8	5	69.68	56.92
12	5	75.81	83.03
13	5	67.36	80.15
16	5	76.86	95.72
18	5	92.69	71.69
22	5	74.49	98.30
24	5	97.14	85.72

## Example 10.16.4

List the number of students from each state, studying each major and in each year of study.

-- Example 10.16.4

```
--      select ST_STATE, ST_MAJOR, ST_YEAR, count(*)
--      from STUDENT
--      group by ST_STATE, ST_MAJOR, ST_YEAR ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.16.4");

DECLAR ( CURSOR , CURSOR\_FOR =>

# UNCLASSIFIED

```

        SELECT ( ST_STATE & ST_MAJOR & ST_YEAR & count ('*'),
        FROM => STUDENT,
        GROUP_BY => ST_STATE & ST_MAJOR & ST_YEAR ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT ("ST_STATE    ST_MAJOR    ST_YEAR    COUNT");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_ST_STATE, V_ST_STATE_INDEX);
        INTO (V_ST_MAJOR);
        INTO (V_ST_YEAR);
        INTO (COUNT_RESULT);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- ST_STATE
        PUT (V_ST_STATE, V_ST_STATE_INDEX);
        SET_COL (12); -- ST_MAJOR
        PUT (V_ST_MAJOR, 1);
        SET_COL (23); -- ST_YEAR
        PUT (V_ST_YEAR);
        SET_COL (33); -- COUNT
        PUT (COUNT_RESULT, 3);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.16.4

ST_STATE	ST_MAJOR	ST_YEAR	COUNT
DC	1	ONE	1
DC	1	TWO	1
DC	3	THREE	1
MD	1	THREE	1
MD	4	ONE	1

## UNCLASSIFIED

MD	4	FOUR	1
MD	5	THREE	1
NC	1	THREE	1
NC	2	TWO	1
NC	4	ONE	1
NY	5	ONE	1
NY	5	FOUR	1
PA	3	FOUR	1
PA	4	TWO	1
PA	4	THREE	1
PA	5	FOUR	1
SC	2	TWO	1
SC	5	TWO	1
VA	1	THREE	1
VA	2	ONE	2
VA	3	THREE	1
VA	3	FOUR	1
VA	5	ONE	1
WV	2	TWO	1

### 10.17 Nested Queries

#### Example 10.17.1

If we wanted to find out which professor made the highest salary, without having to look at the salaries for each one, we'd have to write a query to select the maximum salary from the professor table, such as:

```
-- Example 10.17.1

--      select max (PROF_SALARY)
--      from PROFESSOR ;

begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.17.1");

  SELEC ( max (PROF_SALARY),
    FROM => PROFESSOR ) ;
    INTO ( V_PROF_SALARY );

  NEW_LINE;
  PUT_LINE ("MAX_PROF_SALARY");
  SET_COL (1);  -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
  NEW_LINE;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
```

# UNCLASSIFIED

```

        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

## Output of Example 10.17.1

```

MAX_PROF_SALARY
      50000.00
```

## Example 10.17.2

Then we'd have to write a second query to list the professors who earn \$50,000, which is what we discovered to be the maximum salary in the last query.

```

-- Example 10.17.2

--      select PROF_FIRST, PROF_NAME, PROF_SALARY
--      from PROFESSOR
--      where PROF_SALARY = 50000.00 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.2");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( PROF_FIRST & PROF_NAME & PROF_SALARY,
    FROM => PROFESSOR,
    WHERE => EQ ( PROF_SALARY, 50000.00 ) ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_FIRST  PROF_NAME      PROF_SALARY");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
        INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
        INTO ( V_PROF_SALARY );
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- PROF_FIRST
        PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
        SET_COL (13); -- PROF_NAME
```

UNCLASSIFIED

```
        PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (27); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

**Output of Example 10.17.2**

PROF_FIRST	PROF_NAME	PROF_SALARY
Bruce	Bailey	50000.00

We would have to get the desired information with two queries since we cannot use aggregate functions in the where clause. This would not be as cumbersome with Ada/SQL as with interactive queries. It could be done in one program in Ada/SQL by saving the maximum salary found in the first query in a variable and using that variable for comparison in the second query. However nested queries simplify it even more.

Nested queries are used when you want to select records from a table using selection criteria contained within that same table. In the above example we needed to know the maximum salary before we could select all records with the maximum salary. When nesting queries you use the result of one query as the selection criteria for the next query. A nested query may be a part of the where clause of a query. A nested query is called a subquery of the query in which it is nested. The simplest form of a subquery returns only one value. For example the above subquery returned the maximum salary. The format of a simple one value subquery is:

```
SELECT ( columns,
        FROM => tables,
        WHERE => column_conditions OPERATOR
        SELECT ( column,
                FROM => table,
                WHERE => where_conditions ... ) ) ;
```

A subquery of this format must return only one record or value. If more than one record is selected in the subquery you will get the exception UNHANDELED\_RDBMS\_ERROR at execution time. Queries may be nested to any level with the results of the deepest one being the conditions for the next.

**Example 10.17.3**

To do the above queries as one nested query we would list the professor's names who earn the maximum salary of all professors.

-- Example 10.17.3

```
--      select PROF_FIRST, PROF_NAME, PROF_SALARY
--      from PROFESSOR
--      where PROF_SALARY =
--            ( select max (PROF_SALARY)
--              from PROFESSOR ) ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.17.3");

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_FIRST & PROF_NAME & PROF_SALARY,
        FROM => PROFESSOR,
        WHERE => EQ ( PROF_SALARY,
        SELEC ( max (PROF_SALARY),
        FROM => PROFESSOR ) ) ) ) ;
```

OPEN ( CURSOR );

begin

NEW\_LINE;

PUT\_LINE ("PROF\_FIRST PROF\_NAME PROF\_SALARY");

GOT\_ONE := 0;

loop

```
    FETCH ( CURSOR );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;
```

```
    SET_COL (1); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
```

```
    SET_COL (13); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
```

```
    SET_COL (27); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
```

NEW\_LINE;

end loop;

exception

```
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
```

# UNCLASSIFIED

```

                                end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.17.3

PROF_FIRST	PROF_NAME	PROF_SALARY
Bruce	Bailey	50000.00

Multiple subqueries may be linked together in the where clause by using ANDs and/or ORs.

## Example 10.17.4

For example list professor id and salary for all professors who are not earning the minimum or the maximum salary.

```

-- Example 10.17.4

--      select PROF_ID, PROF_SALARY
--      from PROFESSOR
--      where PROF_SALARY >
--            ( select min ( PROF_SALARY )
--              from PROFESSOR )
--      and PROF_SALARY <
--            ( select max ( PROF_SALARY )
--              from PROFESSOR ) ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.17.4");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( PROF_ID & PROF_SALARY,
    FROM => PROFESSOR,
    WHERE => PROF_SALARY >
      SELEC ( min ( PROF_SALARY ),
        FROM => PROFESSOR )
    AND PROF_SALARY <
      SELEC ( max ( PROF_SALARY ),
        FROM => PROFESSOR ) ) ) ;

```

```

OPEN ( CURSOR );

```

```

begin
  NEW_LINE;
  PUT_LINE ("PROF_ID  PROF_SALARY");

```

# UNCLASSIFIED

```

GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR      => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.17.4

PROF_ID	PROF_SALARY
1	35000 00
2	45000.00
5	40000.00

## Example 10.17.5

A subquery may also have a subquery of it's own. For example list the id, salary and number of years of employment for the professors earning more than the minimum salary of professors who have served more than the average number of years.

```

-- Example 10.17.5

--      select PROF_ID, PROF_SALARY, PROF_YEARS
--      from PROFESSOR
--      where PROF_SALARY >
--            ( select min ( PROF_SALARY )
--              from PROFESSOR
--              where PROF_YEARS >
--                ( select avg ( PROF_YEARS )

```



UNCLASSIFIED

```
--
                                from PROFESSOR ) ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.5");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_ID & PROF_SALARY & PROF_YEARS,
        FROM => PROFESSOR,
        WHERE => PROF_SALARY >
        SELEC ( min ( PROF_SALARY ),
        FROM => PROFESSOR,
        WHERE => PROF_YEARS >
        SELEC ( avg ( PROF_YEARS ),
        FROM => PROFESSOR ) ) ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_SALARY  PROF_YEARS");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_SALARY );
    INTO ( V_PROF_YEARS );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    SET_COL (23); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

## Output of Example 10.17.5

PROF_ID	PROF_SALARY	PROF_YEARS
4	50000.00	15

Subqueries may also be written to return a set of values instead of only one value. The where clause must specify how the values returned are to be treated. This is specified by using the keyword ANY or ALLL between the comparison operator and the subquery in the where clause and enclosing the subquery in an additional set of parenthesis. When using the keyword ANY, if the comparison to any of the values selected in the subquery is true then the record is selected. When using the keyword ALLL, the comparison to each of the values selected in the subquery must be true in order for the record to be selected. ALLL with an extra L is used as the Ada/SQL keyword since Ada reserves ALL. The format of a multi value subquery is:

```

SELEC ( columns,
      FROM => tables,
      WHERE => column_conditions OPERATOR ANY/ALLL
      ( SELEC ( column,
              FROM => table,
              WHERE => where_conditions ... ) ) ) ;

```

## Example 10.17.6

For example select any student who is taking more than two classes. List the student's id and average grade.

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
  FROM => CLASS,
  WHERE => EQ ( CLASS_STUDENT, ANY
    ( SELEC ( CLASS_STUDENT,
      FROM => CLASS,
      GROUP_BY => CLASS_STUDENT,
      HAVING => count ( '*' ) > 2 ) ) ) ) ;

```

The above would be the query of choice. However if you attempt to scan this query you will get the error "%ADASQL-E-SCAN, Identifier has no valid meaning in this context". This is because the ANY and ALLL functions have not been implemented in Level 1 of Ada/SQL. We can replace the ANY function with the IS\_IN function with the same results. Unfortunately there is no replacement for the ALLL function in Level 1.

-- Example 10.17.6

```

--      select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS
--      where CLASS_STUDENT = any
--      ( select CLASS_STUDENT
--        from CLASS
--        group by CLASS_STUDENT
--        having count ( * ) > 2 ) ;

```

# UNCLASSIFIED

```

NEW_LINE;
PUT_LINE ("Output of Example 10.17.6");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
  FROM => CLASS,
  WHERE => IS_IN ( CLASS_STUDENT,
  SELEC ( CLASS_STUDENT,
  FROM => CLASS,
  GROUP_BY => CLASS_STUDENT,
  HAVING => count ( '*' ) > 2 ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("CLASS_STUDENT    CLASS_SEM_1 + CLASS_SEM_2 / 2");
GOT_ONE := 0;

loop
  FETCH ( CURSOR );
  INTO (V_CLASS_STUDENT);
  INTO (AVG_SEM_1);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- CLASS_STUDENT
  PUT (V_CLASS_STUDENT, 3);
  SET_COL (17); -- AVG_SEM_1
  PUT (AVG_SEM_1, 3, 2, 0);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.17.6

CLASS_STUDENT	CLASS_SEM_1 + CLASS_SEM_2 / 2
7	100.00
7	100.00

# UNCLASSIFIED

7	100.00
7	100.00
16	81.95
16	93.66
16	86.29
16	84.63
16	89.14
16	93.06
22	72.78
22	86.39
22	87.36

## Example 10.17.7

Select all the students and their average grade for all classes where the student's grade is greater than or equal to all grades earned by all students in all classes. In other words select the highest grades earned. Following is the query, however we cannot run it since ALLL is not currently implemented.

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) ,
    FROM => CLASS,
    WHERE => ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 >= ALLL
      ( SELEC ( ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) ,
        FROM => CLASS ) ) ) ;
```

## Example 10.17.8

You must be careful when deciding if you should use the keyword ANY or ALLL. For example if we were to select students equal to ALLL the students taking more than two classes we would select no records. Again the query would be as follows but we cannot run it.

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) ,
    FROM => CLASS,
    WHERE => EQ ( CLASS_STUDENT, ALLL
      ( SELEC ( CLASS_STUDENT,
        FROM => CLASS,
        GROUP_BY => CLASS_STUDENT,
        HAVING => count ( '*' ) > 2 ) ) ) ) ;
```

Why would we get no records? Because our column CLASS\_STUDENT will be compared to all of the CLASS\_STUDENT columns from the subquery and must be equal to all of them in order to be selected. If more than one student is taking more than two classes the column in the record which we are checking cannot be equal to all values for the column selected in the subquery. Therefore every record is rejected.

**Example 10.17.9**

Likewise if we were to select the class grades where the average grade was greater than or equal to any other grade in the table we'd end up selecting all records. Following is the query. Any is not implemented and in this case we cannot substitute IS\_IN for ANY, so we will be unable to run this query.

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS,
    WHERE => ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 >= ANY
      ( SELEC ( ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
        FROM => CLASS ) ) ) ) ;

```

We would have selected every record since we are comparing the column on the current record to the same column on every record in the table and of course it will be greater than or equal to at least one other record in the table.

**Example 10.17.10**

List the average grades for all classes taken by any student taking more than two classes and where the student's average grade is at least as high as the overall student average for students taking at least three classes. We have to substitute IS\_IN for ANY in this query to run it. Following is the query with ANY and then the code we're substituting it with.

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS,
    WHERE => ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 >=
      SELEC ( avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
        FROM => CLASS,
        WHERE => EQ ( CLASS_STUDENT, ANY
          ( SELEC ( CLASS_STUDENT,
            FROM => CLASS,
            GROUP_BY => CLASS_STUDENT,
            HAVING => count ( '*' ) > 2 ) ) ) ) )
    AND EQ ( CLASS_STUDENT, ANY
      ( SELEC ( CLASS_STUDENT,
        FROM => CLASS,
        GROUP_BY => CLASS_STUDENT,
        HAVING => count ( '*' ) > 2 ) ) ) ) ) ;

```

-- Example 10.17.10

```

--      select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS
--      where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 >=
--        ( select avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--          from CLASS
--          where CLASS_STUDENT = any

```

# UNCLASSIFIED

```

--      ( select CLASS_STUDENT
--      from CLASS
--      group by CLASS_STUDENT
--      having count (*) > 2 ) )
-- and CLASS_STUDENT = any
--      ( select CLASS_STUDENT
--      from CLASS
--      group by CLASS_STUDENT
--      having count (*) > 2 ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.10");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
  FROM => CLASS,
  WHERE => ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 >=
  SELEC ( avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
  FROM => CLASS,
  WHERE => IS_IN ( CLASS_STUDENT,
  SELEC ( CLASS_STUDENT,
  FROM => CLASS,
  GROUP_BY => CLASS_STUDENT,
  HAVING => count ( '*' ) > 2 ) ) )
  AND IS_IN ( CLASS_STUDENT,
  SELEC ( CLASS_STUDENT,
  FROM => CLASS,
  GROUP_BY => CLASS_STUDENT,
  HAVING => count ( '*' ) > 2 ) ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_STUDENT  CLASS_SEM_1 + CLASS_SEM_2 / 2");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (AVG_SEM_1);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (17); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    NEW_LINE;
  end loop;

exception

```

# UNCLASSIFIED

```

when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.17.10

CLASS_STUDENT	CLASS_SEM_1 + CLASS_SEM_2 / 2
7	100.00
7	100.00
7	100.00
7	100.00
16	93.66
16	93.06

We've already demonstrated the use of IS\_IN in subqueries as substitution for ANY. You can also use the operator NOT IS\_IN with a subquery. Use IS\_IN when you wish to select records which match a record in the list of selected records from the subquery. Use NOT IS\_IN when you wish to select records which do not match any record in the list of selected records from the subquery.

## Example 10.17.11

For example list any student and his average grade from a list of students who are taking more than two classes.

```

-- Example 10.17.11

--      select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS
--      where CLASS_STUDENT in
--      ( select CLASS_STUDENT
--        from CLASS
--        group by CLASS_STUDENT
--        having count (*) > 2 ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.11");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS,
    WHERE => IS_IN ( CLASS_STUDENT,

```

# UNCLASSIFIED

```

        SELEC ( CLASS_STUDENT,
                FROM => CLASS,
                GROUP_BY => CLASS_STUDENT,
                HAVING => count ( '*' ) > 2 ) ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ( "CLASS_STUDENT    CLASS_SEM_1 + CLASS_SEM_2 / 2" );
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO ( V_CLASS_STUDENT );
        INTO ( AVG_SEM_1 );
        GOT_ONE := GOT_ONE + 1;

        SET_COL ( 1 ); -- CLASS_STUDENT
        PUT ( V_CLASS_STUDENT, 3 );
        SET_COL ( 17 ); -- AVG_SEM_1
        PUT ( AVG_SEM_1, 3, 2, 0 );
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ( "EXCEPTION: Not Found Error" );
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ( "EXCEPTION: No Update Error" );
    when UNIQUE_ERROR    => PUT_LINE ( "EXCEPTION: Unique Error" );
end;

CLOSE ( CURSOR );

```

## Output of Example 10.17.11

CLASS_STUDENT	CLASS_SEM_1 + CLASS_SEM_2 / 2
7	100.00
7	100.00
7	100.00
7	100.00
16	81.95
16	93.66
16	86.29
16	84.63
16	89.14
16	93.06



# UNCLASSIFIED

22	72.78
22	86.39
22	87.36

## Example 10.17.12

We could also list any student and his average grade from a list of students who are not taking fewer than two classes.

-- Example 10.17.12

```
--      select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS
--      where CLASS_STUDENT not in
--      ( select CLASS_STUDENT
--        from CLASS
--        group by CLASS_STUDENT
--        having count (*) <= 2 ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.12");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00),
  FROM => CLASS,
  WHERE => NOT IS_IN ( CLASS_STUDENT,
    SELEC ( CLASS_STUDENT,
      FROM => CLASS,
      GROUP_BY => CLASS_STUDENT,
      HAVING => count ('*') <= 2 ) ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_STUDENT   CLASS_SEM_1 + CLASS_SEM_2 / 2");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (AVG_SEM_1);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (17); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    NEW_LINE;
```

## UNCLASSIFIED

```
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.17.12

CLASS_STUDENT	CLASS_SEM_1 + CLASS_SEM_2 / 2
7	100.00
7	100.00
7	100.00
7	100.00
16	81.95
16	93.06
16	89.14
16	84.63
16	93.66
16	86.29
22	72.78
22	87.36
22	86.39

### 10.18 HAVING

The GROUP\_BY clause allows you to group your records together based on a common element. A WHERE clause is used only to select or reject individual records. It cannot be used to select or reject entire groups of records. A WHERE clause cannot use aggregate functions for comparison since they relate to entire groups of records and not individual records. The HAVING clause allows you to select or reject an entire group formed by the GROUP\_BY clause and to use aggregate functions for comparison. The HAVING clause must always be used with a GROUP\_BY clause. A query may contain both a WHERE clause and a HAVING clause in which case the WHERE clause is used to select the individual records which will make up the groups and the HAVING clause is used to select the groups. A HAVING clause may contain a nested query. The format of the HAVING clause in a SELECT query is:

```
SELECT ( columns,
        FROM => tables,
        GROUP_BY => columns,
        HAVING => column_comparison_clause ) ;
```

**Example 10.18.1**

List the departments having more than ten class hours.

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( COURSE_DEPT & sum ( COURSE_HOURS ),
    FROM => COURSE,
    GROUP_BY => COURSE_DEPT,
    HAVING => sum ( COURSE_HOURS ) > TEN ) ) ;

```

The above statement would be used for this query. We have defined COURSE\_HOURS as an enumeration type for this program. Therefore in the HAVING clause we must use TEN instead of 10 to avoid an error in the application scanner. This query will get a constraint error on execution because of the enumeration type. You simply cannot preform the aggregate function sum or avg on an enumeration type.

Just to show an example of the having clause with the aggregate function sum we will replace COURSE\_HOURS with COURSE\_PROF which is a numeric field. This query makes no sense but it will run.

-- Example 10.18.1

```

--      select COURSE_DEPT, sum ( COURSE_PROF)
--      from COURSE
--      group by COURSE_DEPT
--      having sum ( COURSE_PROF ) > 10 ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.18.1");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( COURSE_DEPT & sum ( COURSE_PROF ),
    FROM => COURSE,
    GROUP_BY => COURSE_DEPT,
    HAVING => sum ( COURSE_PROF ) > 10 ) ) ;

```

```

OPEN ( CURSOR );

```

```

begin

```

```

  NEW_LINE;
  PUT_LINE ("COURSE_DEPT    COURSE_PROF ");
  GOT_ONE := 0;

```

```

  loop

```

```

    FETCH ( CURSOR );
    INTO (V_COURSE_DEPT);
    INTO (V_COURSE_PROF);
    GOT_ONE := GOT_ONE + 1;

```

```

  SET_COL (1);  -- COURSE_DEPT
  PUT (V_COURSE_DEPT, 1);

```

# UNCLASSIFIED

```

SET_COL (15); -- COURSE_PROF
PUT (V_COURSE_PROF, 3);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.18.1

COURSE_DEPT	COURSE_PROF
1	15
2	12
5	13

## Example 10.18.2

List all the departments with more than three classes.

```

-- Example 10.18.2

--      select COURSE_DEPT, count (*)
--      from COURSE
--      group by COURSE_DEPT
--      having count (*) > 3 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.18.2");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( COURSE_DEPT & count ('*'),
    FROM => COURSE,
    GROUP_BY => COURSE_DEPT,
    HAVING => count ('*') > 3 ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("COURSE_DEPT    COUNT");

```

# UNCLASSIFIED

```
GOT_ONE := 0;

loop
  FETCH ( CURSOR );
  INTO (V_COURSE_DEPT);
  INTO (COUNT_RESULT);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- COURSE_DEPT
  PUT (V_COURSE_DEPT, 1);
  SET_COL (15); -- COUNT_RESULT
  PUT (COUNT_RESULT, 3);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

## Output of Example 10.18.2

```
COURSE_DEPT  COUNT
           2      4
```

## Example 10.18.3

List the number of students from Virginia, District of Columbia, Maryland, North Carolina and Pennsylvania grouping them by their majors within their home states, but list only those groups having an average number of student years greater than two.

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( ST_STATE & ST_MAJOR & count ('*'),
    FROM => STUDENT,
    WHERE => IS_IN ( ST_STATE, "VA" or "DC" or "MD" or "NC" or "PA" ),
    GROUP_BY => ST_STATE & ST_MAJOR,
    HAVING => avg ( ST_YEAR ) > TWO ) ) ;
```

This would be the statement to preform the query. ST\_YEAR is an enumeration type. This query cannot be run without causing an exception. Here is the same query replacing ST\_YEAR with ST\_MAJOR, a numeric field. Not a very useful query but it serves as an example.

# UNCLASSIFIED

```
-- Example 10.18.3

--      select ST_STATE, ST_MAJOR, count (*)
--      from STUDENT
--      where ST_STATE in ( 'VA', 'DC', 'MD', 'NC', 'PA' )
--      group by ST_STATE, ST_MAJOR
--      having avg ( ST_MAJOR ) > 2 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.18.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_STATE & ST_MAJOR & count ('*'),
        FROM => STUDENT,
        WHERE => IS_IN ( ST_STATE, "VA" or "DC" or "MD" or "NC" or "PA" ),
        GROUP_BY => ST_STATE & ST_MAJOR,
        HAVING => avg ( ST_MAJOR ) > 2 ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_STATE ST_MAJOR COUNT");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (COUNT_RESULT);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (11); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (21); -- COUNT_RESULT
    PUT (COUNT_RESULT, 3);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

# UNCLASSIFIED

```
CLOSE ( CURSOR );
```

## Output of Example 10.18.3

ST_STATE	ST_MAJOR	COUNT
DC	3	1
MD	4	2
MD	5	1
NC	4	1
PA	3	1
PA	4	2
PA	5	1
VA	3	2
VA	5	1

## Example 10.18.4

List classes taken in department two and four in which the student grades were above the average for all classes.

```
-- Example 10.18.4
```

```
--      select CLASS_COURSE, avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--      from CLASS
--      where CLASS_DEPT = 2 or CLASS_DEPT = 4
--      group by CLASS_COURSE
--      having avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 ) >
--      ( select avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--        from CLASS ) ;
```

```
NEW_LINE;
```

```
PUT_LINE ("Output of Example 10.18.4");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_COURSE & avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS,
    WHERE => EQ ( CLASS_DEPT, 2 )
    OR      EQ ( CLASS_DEPT, 4 ),
    GROUP_BY => CLASS_COURSE,
    HAVING => avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) >
    SELEC ( avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
      FROM => CLASS ) ) ) ;
```

```
OPEN ( CURSOR );
```

```
begin
```

```
NEW_LINE;
```

```
PUT_LINE ("CLASS_COURSE  AVG CLASS_SEM_1 + CLASS_SEM_2 / 2");
```

```
GOT_ONE := 0;
```

# UNCLASSIFIED

```

loop
  FETCH ( CURSOR );
  INTO (V_CLASS_COURSE);
  INTO (AVG_SEM_1);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- CLASS_COURSE
  PUT (V_CLASS_COURSE, 3);
  SET_COL (15); -- AVG_SEM_1
  PUT (AVG_SEM_1, 3, 2, 0);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_      97.14      85.72

```

## Example 10.16.4

List the number of students from each state, studying each major and in each year of study.

```

-- Example 10.16.4

--      select ST_STATE, ST_MAJOR, ST_YEAR, count(*)
--      from STUDENT
--      group by ST_STATE, ST_MAJOR, ST_YEAR ;

NEW_LINE;
PUT_LINE ("Example 10.16.4");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( ST_STATE & ST_MAJOR & ST_YEAR & count ('*'),
    FROM => STUDENT,
    GROUP_BY => ST_STATE & ST_MAJOR & S402      96.31
403      96.29

```

## Example 10.18.5

List the average semester grades for the classes which have the students with the highest and the lowest average class grade.

```

-- Example 10.18.5

--      select CLASS_COURSE, avg ( CLASS_SEM_1 ), avg ( CLASS_SEM_2 )
--      from CLASS
--      group by CLASS_COURSE
--      having CLASS_COURSE =
--      ( select CLASS_COURSE
--      from CLASS
--      where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =

```



# UNCLASSIFIED

```

--          ( select max ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--            from CLASS )
--          or ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--          ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--            from CLASS ) ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.18.5");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_COURSE & avg ( CLASS_SEM_1 ) & avg ( CLASS_SEM_2 ),
    FROM => CLASS,
    GROUP_BY => CLASS_COURSE,
    HAVING => EQ ( CLASS_COURSE,
      SELEC ( CLASS_COURSE,
        FROM => CLASS,
        WHERE => EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
          SELEC ( max ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
            FROM => CLASS ) )
        OR EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
          SELEC ( min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
            FROM => CLASS ) ) ) ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_COURSE   AVG SEM_1   AVG SEM_2");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_COURSE);
    INTO (AVG_SEM_1);
    INTO (AVG_SEM_2);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_COURSE
    PUT (V_CLASS_COURSE, 3);
    SET_COL (16); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    SET_COL (28); -- AVG_SEM_2
    PUT (AVG_SEM_2, 3, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else

```

# UNCLASSIFIED

```

        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

This query results in the exception UNHANDLED\_RDBMS\_ERROR because the highest and/or lowest student grades are shared by more than one student. We will have to use the IS\_IN qualifier for the subquery. In the interactive example we used ANY, but here we will use IS\_IN. Let's try again.

## Example 10.18.6

This time list the average semester grades for the classes which have the students with the highest and the lowest average class grade and remember to use the IS\_IN qualifier.

--Example 10.18.6

```

--      select CLASS_COURSE, avg ( CLASS_SEM_1 ), avg ( CLASS_SEM_2 )
--      from CLASS
--      group by CLASS_COURSE
--      having CLASS_COURSE = any
--          ( select CLASS_COURSE
--            from CLASS
--            where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--                  ( select max ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--                    from CLASS )
--            or ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--                  ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--                    from CLASS ) ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.18.6");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( CLASS_COURSE & avg ( CLASS_SEM_1 ) & avg ( CLASS_SEM_2 ),
    FROM => CLASS,
    GROUP_BY => CLASS_COURSE,
    HAVING => IS_IN ( CLASS_COURSE,
    SELEC ( CLASS_COURSE,
    FROM => CLASS,
    WHERE => EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
    SELEC ( max ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS ) )
    OR      EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
    SELEC ( min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS ) ) ) ) ) ;

OPEN ( CURSOR );

```

# UNCLASSIFIED

```

begin
  NEW_LINE;
  PUT_LINE ("CLASS_COURSE   AVG SEM_1   AVG SEM_2");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_COURSE);
    INTO (AVG_SEM_1);
    INTO (AVG_SEM_2);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_COURSE
    PUT (V_CLASS_COURSE, 3);
    SET_COL (16); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    SET_COL (28); -- AVG_SEM_2
    PUT (AVG_SEM_2, 3, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.18.6

CLASS_COURSE	AVG SEM_1	AVG SEM_2
401	92.23	79.88
402	98.78	93.84
403	94.28	98.29
502	68.52	68.54
503	90.63	87.37

## Example 10.18.7

List the students and their second semester grades for the class which has the highest average grade of all students in the second semester.

--Example 10.18.7

UNCLASSIFIED

```
--      select CLASS_STUDENT, CLASS_SEM_2
--      from CLASS
--      where CLASS_COURSE =
--      ( select CLASS_COURSE
--      from CLASS
--      group by CLASS_COURSE
--      having avg ( CLASS_SEM_2 ) =
--      ( select max ( avg ( CLASS_SEM_2 ) )
--      from CLASS
--      group by CLASS_COURSE ) ) ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.18.7");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & CLASS_SEM_2,
    FROM => CLASS,
    WHERE => EQ ( CLASS_COURSE,
      SELEC ( CLASS_COURSE,
        FROM => CLASS,
        GROUP_BY => CLASS_COURSE,
        HAVING => EQ ( avg ( CLASS_SEM_2 ),
          SELEC ( max ( avg ( CLASS_SEM_2 ) ),
            FROM => CLASS,
            GROUP_BY => CLASS_COURSE ) ) ) ) ) ) ;
```

```
OPEN ( CURSOR );
```

```
begin
  NEW_LINE;
  PUT_LINE ("CLASS_STUDENT  CLASS_SEM_2");
  GOT_ONE := 0;
```

```
  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (V_CLASS_SEM_2);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (17); -- CLASS_SEM_2
    PUT (V_CLASS_SEM_2, 3, 2, 0);
    NEW_LINE;
  end loop;
```

```
exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
```

## UNCLASSIFIED

```
                null;
            end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.18.7

CLASS_STUDENT	CLASS_SEM_2
3	97.48
7	100.00
16	97.40

### 10.19 Joining Multiple Tables

So far we've only been able to select data from one table in each query. Frequently you will wish to use the data from two or more tables in the same query. For example when we have listed the student while selecting from the class table we get the student's id number, not his name. Normally we'd want to list the name from the student table and the grading information from the class table. We would do this by listing more than one table in the from clause. This is called joining tables. The format of a query joining tables is:

```
SELECT ( column & column & ...
        FROM => table & table & ...
        ... .. ) ;
```

All clauses that may be used with a select, such as where, order by, group by, having and nested queries, may also be used when joining tables. These clauses will be used to specify how the tables should be joined together. Generally you will specify the join in the where clause by having a statement that links columns from two tables together. If you do not specify how to join the tables you will get a list of every entry in each table joined together.

#### Example 10.19.1

For example list all columns in the department and the professor table in a joined query.

--Example 10.19.1

```
--      select *
--      from DEPARTMENT, PROFESSOR ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.19.1");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*' ,
```

UNCLASSIFIED

```

FROM => DEPARTMENT & PROFESSOR ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("DEPARTMENT      PROFESSOR");
PUT_LINE ("ID   DESC      ID   NAME      FIRST      DEPT   YEARS   SALARY");
GOT_ONE := 0;

loop
  FETCH ( CURSOR );
  INTO ( V_DEPT_ID );
  INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
  INTO ( V_PROF_ID );
  INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
  INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
  INTO ( V_PROF_DEPT );
  INTO ( V_PROF_YEARS );
  INTO ( V_PROF_SALARY );
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- DEPT_ID
  PUT (V_DEPT_ID, 1);
  SET_COL (6); -- DEPT_DESC
  PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
  SET_COL (16); -- PROF_ID
  PUT (V_PROF_ID, 2);
  SET_COL (20); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
  SET_COL (34); -- PROF_FIRST
  PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
  SET_COL (46); -- PROF_DEPT
  PUT (V_PROF_DEPT, 1);
  SET_COL (52); -- PROF_YEARS
  PUT (V_PROF_YEARS, 2);
  SET_COL (59); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

## UNCLASSIFIED

```
CLOSE ( CURSOR );
```

## Output of Example 10.19.1

DEPARTMENT		PROFESSOR					
ID	DESC	ID	NAME	FIRST	DEPT	YEARS	SALARY
1	History	1	Dysart	Gregory	3	3	35000.00
2	Math	1	Dysart	Gregory	3	3	35000.00
3	Science	1	Dysart	Gregory	3	3	35000.00
4	Language	1	Dysart	Gregory	3	3	35000.00
5	Art	1	Dysart	Gregory	3	3	35000.00
1	History	2	Hall	Elizabeth	4	7	45000.00
2	Math	2	Hall	Elizabeth	4	7	45000.00
3	Science	2	Hall	Elizabeth	4	7	45000.00
4	Language	2	Hall	Elizabeth	4	7	45000.00
5	Art	2	Hall	Elizabeth	4	7	45000.00
1	History	3	Steinbacner	Moris	2	1	30000.00
2	Math	3	Steinbacner	Moris	2	1	30000.00
3	Science	3	Steinbacner	Moris	2	1	30000.00
4	Language	3	Steinbacner	Moris	2	1	30000.00
5	Art	3	Steinbacner	Moris	2	1	30000.00
1	History	4	Bailey	Bruce	5	15	50000.00
2	Math	4	Bailey	Bruce	5	15	50000.00
3	Science	4	Bailey	Bruce	5	15	50000.00
4	Language	4	Bailey	Bruce	5	15	50000.00
5	Art	4	Bailey	Bruce	5	15	50000.00
1	History	5	Clements	Carol	1	4	40000.00
2	Math	5	Clements	Carol	1	4	40000.00
3	Science	5	Clements	Carol	1	4	40000.00
4	Language	5	Clements	Carol	1	4	40000.00
5	Art	5	Clements	Carol	1	4	40000.00

You can see how every entry in the department table is joined with each entry in the professor table. Joining output in this way is really quite useless. However joins are very useful when used correctly. The professor table lists a department for each professor. But only the department id is included in the professor table, so by looking at records selected from the professor table we see a number in the column for department. This is pretty useless unless we then cross reference the department id in the department table to find the description of the department assigned to the professor. Remember why we used the id in the professor record instead of the complete description. It was to minimize the information stored in each table and to allow us to point to the detailed information.

## Example 10.19.2

Let's join the department and professor table to produce a list of the professors and a description of the department assigned to them. We will do this by joining on the prof\_dept column of the professor table and the dept\_id column of the department table.

```
--Example 10.19.2
```

# UNCLASSIFIED

```
--      select PROF_FIRST, PROF_NAME, DEPT_DESC
--      from PROFESSOR, DEPARTMENT
--      where PROF_DEPT = DEPT_ID ;

NEW_LINE;
PUT_LINE ("Output of Example 10.19.2");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_FIRST & PROF_NAME & DEPT_DESC,
        FROM => PROFESSOR & DEPARTMENT,
        WHERE => EQ ( PROF_DEPT, DEPT_ID ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_FIRST      PROF_NAME      DEPT_DESC");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_FIRST
    PUT ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    SET_COL (14); -- PROF_NAME
    PUT ( V_PROF_NAME, V_PROF_NAME_INDEX );
    SET_COL (28); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

## Output of Example 10.19.2

PROF_FIRST	PROF_NAME	DEPT_DESC
------------	-----------	-----------



# UNCLASSIFIED

Carol	Clements	History
Moris	Steinbacner	Math
Gregory	Dysart	Science
Elizabeth	Hall	Language
Bruce	Bailey	Art

## Example 10.19.3

List the description of the department and the course and the professor's first and last name for each course offered.

--Example 10.19.3

```
--      select DEPT_DESC, COURSE_DESC, PROF_FIRST, PROF_NAME
--      from PROFESSOR, DEPARTMENT, COURSE
--      where COURSE_DEPT = DEPT_ID
--      and COURSE_PROF = PROF_ID ;

NEW_LINE;
PUT_LINE ("Output of Example 10.19.3");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( DEPT_DESC & COURSE_DESC & PROF_FIRST & PROF_NAME,
    FROM => PROFESSOR & DEPARTMENT & COURSE,
    WHERE => EQ ( COURSE_DEPT, DEPT_ID )
      AND      EQ ( COURSE_PROF, PROF_ID ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("DEPT_DESC  COURSE_DESC                PROF_FIRST  PROF_NAME");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1);  -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    SET_COL (12); -- COURSE_DESC
    PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    SET_COL (34); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (46); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
```

# UNCLASSIFIED

```

NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.19.3

DEPT_DESC	COURSE_DESC	PROF_FIRST	PROF_NAME
Science	Chemistry	Gregory	Dysart
Science	Biology	Gregory	Dysart
Science	Physics	Gregory	Dysart
Language	French	Elizabeth	Hall
Language	Russian	Elizabeth	Hall
Math	Algebra	Moris	Steinbacner
Math	Calculus	Moris	Steinbacner
Math	Trigonometry	Moris	Steinbacner
Math	Geometry	Moris	Steinbacner
Art	Sculpture	Bruce	Bailey
Art	Music	Bruce	Bailey
History	World History	Carol	Clements
Art	Dance	Carol	Clements
History	Political History	Carol	Clements
Language	Spanish	Carol	Clements
History	Ancient History	Carol	Clements

## Example 10.19.4

Again list the description of the department and the course and the professor's first and last name for each course offered but this time order it by the department id and the course id. We do not wish to print department id or course id data. However in order to order by them we must include them in the selection list.

--Example 10.19.4

```

--      select DEPT_DESC, COURSE_DESC, PROF_FIRST, PROF_NAME
--      from PROFESSOR, DEPARTMENT, COURSE
--      where COURSE_DEPT = DEPT_ID
--            and COURSE_PROF = PROF_ID
--            order by DEPT_ID, COURSE_ID ;

```

# UNCLASSIFIED

```

NEW_LINE;
PUT_LINE ("Output of Example 10.19.4");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( DEPT_DESC & COURSE_DESC & PROF_FIRST & PROF_NAME &
    DEPT_ID & COURSE_ID ,
  FROM => PROFESSOR & DEPARTMENT & COURSE,
  WHERE => EQ ( COURSE_DEPT, DEPT_ID )
    AND EQ ( COURSE_PROF, PROF_ID ) ,
  ORDER_BY => DEPT_ID & COURSE_ID ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("DEPT_DESC  COURSE_DESC          PROF_FIRST  PROF_NAME");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    GOT_ONE = GOT_ONE + 1;

    SET_COL (1); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    SET_COL (12); -- COURSE_DESC
    PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    SET_COL (34); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (46); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

Output of Example 10.19.4

# UNCLASSIFIED

DEPT_DESC	COURSE_DESC	PROF_FIRST	PROF_NAME
History	World History	Carol	Clements
History	Political History	Carol	Clements
History	Ancient History	Carol	Clements
Math	Algebra	Moris	Steinbacner
Math	Geometry	Moris	Steinbacner
Math	Trigonometry	Moris	Steinbacner
Math	Calculus	Moris	Steinbacner
Science	Chemistry	Gregory	Dysart
Science	Physics	Gregory	Dysart
Science	Biology	Gregory	Dysart
Language	French	Elizabeth	Hall
Language	Spanish	Carol	Clements
Language	Russian	Elizabeth	Hall
Art	Sculpture	Bruce	Bailey
Art	Music	Bruce	Bailey
Art	Dance	Carol	Clements

## Example 10.19.5

And once again list the description of the department and the course and the professor's first and last name for each course offered but this time order it alphabetically by department and course.

--Example 10.19.5

```
--      select DEPT_DESC, COURSE_DESC, PROF_FIRST, PROF_NAME
--      from PROFESSOR, DEPARTMENT, COURSE
--      where COURSE_DEPT = DEPT_ID
--      and COURSE_PROF = PROF_ID
--      order by DEPT_DESC, COURSE_DESC ;

NEW_LINE;
PUT_LINE ("Output of Example 10.19.5");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( DEPT_DESC & COURSE_DESC & PROF_FIRST & PROF_NAME,
    FROM => PROFESSOR & DEPARTMENT & COURSE,
    WHERE => EQ ( COURSE_DEPT, DEPT_ID )
    AND      EQ ( COURSE_PROF, PROF_ID ) ),
  ORDER_BY => DEPT_DESC & COURSE_DESC ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("DEPT_DESC  COURSE_DESC                PROF_FIRST  PROF_NAME");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
```

# UNCLASSIFIED

```

    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    SET_COL (12); -- COURSE_DESC
    PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    SET_COL (34); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (46); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.19.5

DEPT_DESC	COURSE_DESC	PROF_FIRST	PROF_NAME
Art	Dance	Carol	Clements
Art	Music	Bruce	Bailey
Art	Sculpture	Bruce	Bailey
History	Ancient History	Carol	Clements
History	Political History	Carol	Clements
History	World History	Carol	Clements
Language	French	Elizabeth	Hall
Language	Russian	Elizabeth	Hall
Language	Spanish	Carol	Clements
Math	Algebra	Moris	Steinbacner
Math	Calculus	Moris	Steinbacner
Math	Geometry	Moris	Steinbacner
Math	Trigonometry	Moris	Steinbacner
Science	Biology	Gregory	Dysart
Science	Chemistry	Gregory	Dysart
Science	Physics	Gregory	Dysart

# UNCLASSIFIED

## Example 10.19.6

List the department description, course description, professor's last name and student's last name for the class in which a student earned the highest first semester grade the lowest first semester grade, the highest second semester grade and the lowest semester grade. Be sure to list only one record per student/course/department/professor combination. Also sort the list by department, course, professor, student. In the interactive section we used ANY, here we replace it with IS\_IN.

--Example 10.19.6

```
--      select DEPT_DESC, COURSE_DESC, PROF_NAME, ST_NAME
--      from DEPARTMENT, COURSE, PROFESSOR, STUDENT, CLASS
--      where CLASS_STUDENT = any
--          ( select CLASS_STUDENT
--            from CLASS
--            where CLASS_SEM_1 =
--              ( select max ( CLASS_SEM_1 )
--                from CLASS )
--            or CLASS_SEM_1 =
--              ( select min ( CLASS_SEM_1 )
--                from CLASS )
--            or CLASS_SEM_2 =
--              ( select max ( CLASS_SEM_2 )
--                from CLASS )
--            or CLASS_SEM_2 =
--              ( select min ( CLASS_SEM_2 )
--                from CLASS ) )
--      and CLASS_STUDENT = ST_ID
--      and CLASS_DEPT = DEPT_ID
--      and CLASS_COURSE = COURSE_ID
--      and COURSE_PROF = PROF_ID
--      group by ST_NAME, COURSE_DESC, DEPT_DESC, PROF_NAME
--      order by DEPT_DESC, COURSE_DESC, PROF_NAME, ST_NAME ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.19.6");

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( DEPT_DESC & COURSE_DESC & PROF_NAME & ST_NAME,
    FROM => DEPARTMENT & COURSE & PROFESSOR & STUDENT & CLASS,
    WHERE => IS_IN ( CLASS_STUDENT,
      SELEC ( CLASS_STUDENT,
        FROM => CLASS,
        WHERE => EQ ( CLASS_SEM_1,
          SELEC ( max ( CLASS_SEM_1 ) ,
            FROM => CLASS ) )
        OR EQ ( CLASS_SEM_1,
          SELEC ( min ( CLASS_SEM_1 ) ,
            FROM => CLASS ) )
        OR EQ ( CLASS_SEM_2,
```

UNCLASSIFIED

```

        SELEC ( max ( CLASS_SEM_2 ),
                FROM => CLASS ) )
    OR EQ ( CLASS_SEM_2,
        SELEC ( min ( CLASS_SEM_2 ),
                FROM => CLASS ) ) )
    AND EQ ( CLASS_STUDENT, ST_ID )
    AND EQ ( CLASS_DEPT, DEPT_ID )
    AND EQ ( CLASS_COURSE, COURSE_ID )
    AND EQ ( COURSE_PROF, PROF_ID ),
    GROUP_BY => ST_NAME & COURSE_DESC & DEPT_DESC & PROF_NAME },
    ORDER_BY => DEPT_DESC & COURSE_DESC & PROF_NAME & ST_NAME ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ( "DEPT_DESC    COURSE_DESC                PROF_NAME    ST_NAME" );
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_DEPT_DESC, V_DEPT_DESC_INDEX);
        INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
        INTO (V_PROF_NAME, V_PROF_NAME_INDEX);
        INTO (V_ST_NAME, V_ST_NAME_INDEX);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- DEPT_DESC
        PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
        SET_COL (13); -- COURSE_DESC
        PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
        SET_COL (35); -- PROF_NAME
        PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
        SET_COL (49); -- ST_NAME
        PUT (V_ST_NAME, V_ST_NAME_INDEX);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

# UNCLASSIFIED

## Output of Example 10.19.6

DEPT_DESC	COURSE_DESC	PROF_NAME	ST_NAME
Art	Dance	Clements	Guiffre
History	Ancient History	Clements	McGinn
Language	French	Hall	Guiffre
Language	Russian	Hall	Guiffre
Language	Spanish	Clements	Guiffre
Language	Spanish	Clements	Hess
Science	Biology	Dysart	Horrigan
Science	Chemistry	Dysart	Hess
Science	Physics	Dysart	Horrigan

You do not always have to use an equality to join tables together. You may use any comparison operator.

## Example 10.19.7

For example select the professor's name, number of year's employed, salary and the minimum and maximum suggested salary for the number of years employed.

--Example 10.19.7

```
--      select PROF_NAME, PROF_YEARS, SAL_YEAR, SAL_END, PROF_SALARY,
--             SAL_MIN, SAL_MAX
--      from PROFESSOR, SALARY
--      where PROF_YEARS >= SAL_YEAR
--            and PROF_YEARS <= SAL_END ;

NEW_LINE;
PUT_LINE ("Output of Example 10.19.7");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( PROF_NAME & PROF_YEARS & SAL_YEAR & SAL_END & PROF_SALARY &
    SAL_MIN & SAL_MAX,
  FROM => PROFESSOR & SALARY,
  WHERE => PROF_YEARS >= SAL_YEAR
    AND PROF_YEARS <= SAL_END ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("PROF_NAME      PROF_YEARS  SAL_YEAR  SAL_END  PROF_SALARY  " &
    "SAL_MIN  SAL_MAX");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
```



# UNCLASSIFIED

```

    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_YEARS );
    INTO (V_SAL_YEAR);
    INTO (V_SAL_END);
    INTO ( V_PROF_SALARY );
    INTO (V_SAL_MIN);
    INTO (V_SAL_MAX);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (15); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (27); -- SAL_YEAR
    PUT (V_SAL_YEAR, 2);
    SET_COL (37); -- SAL_END
    PUT (V_SAL_END, 2);
    SET_COL (46); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    SET_COL (59); -- SAL_MIN
    PUT (V_SAL_MIN, 5, 2, 0);
    SET_COL (69); -- SAL_MAX
    PUT (V_SAL_MAX, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR     => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.19.7

PROF_NAME	PROF_YEARS	SAL_YEAR	SAL_END	PROF_SALARY	SAL_MIN	SAL_MAX
Steinbacner	1	1	1	30000.00	20000.00	29999.00
Dysart	3	3	3	35000.00	35000.00	39999.00
Clements	4	4	4	40000.00	40000.00	44999.00
Hall	7	6	10	45000.00	50000.00	51999.00
Bailey	15	11	15	50000.00	52000.00	53999.00

## Example 10.19.8

# UNCLASSIFIED

List the professor's name, salary, number of years employed, the year range for the suggested salary and the suggested salary range which the professor's salary falls into.

--Example 10.19.8

```
--      select PROF_NAME, PROF_SALARY, PROF_YEARS, SAL_YEAR, SAL_END,
--             SAL_MIN, SAL_MAX
--      from PROFESSOR, SALARY
--      where PROF_SALARY between SAL_MIN and SAL_MAX ;

NEW_LINE;
PUT_LINE ("Output of Example 10.19.8");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_NAME & PROF_SALARY & PROF_YEARS & SAL_YEAR & SAL_END &
                SAL_MIN & SAL_MAX,
        FROM => PROFESSOR & SALARY,
        WHERE => BETWEEN ( PROF_SALARY, SAL_MIN and SAL_MAX ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_NAME      PROF_SALARY  PROF_YEARS  SAL_YEAR  SAL_END   " &
        "SAL_MIN    SAL_MAX");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_SALARY );
    INTO ( V_PROF_YEARS );
    INTO (V_SAL_YEAR);
    INTO (V_SAL_END);
    INTO (V_SAL_MIN);
    INTO (V_SAL_MAX);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (15); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    SET_COL (28); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (40); -- SAL_YEAR
    PUT (V_SAL_YEAR, 2);
    SET_COL (50); -- SAL_END
    PUT (V_SAL_END, 2);
    SET_COL (59); -- SAL_MIN
    PUT (V_SAL_MIN, 5, 2, 0);
    SET_COL (69); -- SAL_MAX
```

## UNCLASSIFIED

```
        PUT (V_SAL_MAX, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.19.8

PROF_NAME	PROF_SALARY	PROF_YEARS	SAL_YEAR	SAL_END	SAL_MIN	SAL_MAX
Steinbacner	30000.00	1	2	2	30000.00	34999.00
Dysart	35000.00	3	3	3	35000.00	39999.00
Clements	40000.00	4	4	4	40000.00	44999.00
Hall	45000.00	7	5	5	45000.00	49999.00
Bailey	50000.00	15	6	10	50000.00	51999.00

### 10.20 Correlation Names

There are times when you will have to specify a table name for a column to make it clear which column you want. It is possible to have columns in different tables with the same name. We avoided duplicate column names when we set up our tables. There are two ways to specify which table a column is from. The first is to prefix the column name with the table name in the format of:

TABLE\_NAME.COLUMN\_NAME

The second is through use of a correlation name. With this method you assign a name to the table by generating a redeclaration of a package. The format of this statement is:

```
package NEW_NAME is new TABLE_correlation.name ( "NEW_NAME" );
```

Where NEW\_NAME is the new correlation name you've chosen to use for the table and TABLE is the actual table name you defined in the DDL. For example assign D as a correlation name for the DEPARTMENT table with the statement:

```
package D is new DEPARTMENT_CORRELATION.NAME ( "D" );
```

This package redeclaration should appear after the "use TYPES.ADA\_SQL" statement. In the query to use the correlation name already defined you prefix column names with the correlation name in the select and prefix table names with the correlation name in the from. For example selecting from the DEPARTMENT table using the correlation name defined above:

# UNCLASSIFIED

```

SELEC ( D.DEPT_DESC,
FROM => D.DEPARTMENT,
...

```

## Example 10.20.1

Select department description, course description, professor's name and student's name for the student(s) taking four or more classes. Qualify all column names with the table names. This is a nested query, only qualify columns in the outer query. We will replace ANY in this query with IS\_IN.

--Example 10.20.1

```

--      select DEPARTMENT.DEPT_DESC, COURSE.COURSE_DESC, PROFESSOR.PROF_NAME,
--            STUDENT.ST_NAME
--      from DEPARTMENT, COURSE, PROFESSOR, STUDENT, CLASS
--       where STUDENT.ST_ID = any
--            ( select CLASS_STUDENT
--              from CLASS
--              group by CLASS_STUDENT
--              having COUNT (*) >= 4 )
--      and CLASS.CLASS_STUDENT = STUDENT.ST_ID
--      and CLASS.CLASS_COURSE = COURSE.COURSE_ID
--      and COURSE.COURSE_DEPT = DEPARTMENT.DEPT_ID
--      and COURSE.COURSE_PROF = PROFESSOR.PROF_ID ;

NEW_LINE;
PUT_LINE ("Output of Example 10.20.1");

DECLAR ( CURSOR , CURSOR_FOR =>
      SELEC ( DEPARTMENT.DEPT_DESC & COURSE.COURSE_DESC & PROFESSOR.PROF_NAME &
            STUDENT.ST_NAME,
      FROM => DEPARTMENT & COURSE & PROFESSOR & STUDENT & CLASS,
      WHERE => IS_IN ( STUDENT.ST_ID,
            SELEC ( CLASS_STUDENT,
                  FROM => CLASS,
                  GROUP_BY => CLASS_STUDENT,
                  HAVING => COUNT ( '*' ) >= 4 ) )
      AND EQ ( CLASS.CLASS_STUDENT, STUDENT.ST_ID )
      AND EQ ( CLASS.CLASS_COURSE, COURSE.COURSE_ID )
      AND EQ ( COURSE.COURSE_DEPT, DEPARTMENT.DEPT_ID )
      AND EQ ( COURSE.COURSE_PROF, PROFESSOR.PROF_ID ) ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("DEPT_DESC   COURSE_DESC           PROF_NAME       ST_NAME");
GOT_ONE := 0;

```

# UNCLASSIFIED

```

loop
  FETCH ( CURSOR );
  GOT_ONE := GOT_ONE + 1;
  INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
  INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
  INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
  INTO (V_ST_NAME, V_ST_NAME_INDEX);

  SET_COL (1); -- DEPT_DESC
  PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
  SET_COL (12); -- COURSE_DESC
  PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
  SET_COL (34); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
  SET_COL (48); -- ST_NAME
  PUT (V_ST_NAME, V_ST_NAME_INDEX);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.20.1

DEPT_DESC	COURSE_DESC	PROF_NAME	ST_NAME
Language	French	Hall	Guiffre
Language	Russian	Hall	Guiffre
Art	Dance	Clements	Guiffre
Language	Spanish	Clements	Guiffre
History	World History	Clements	Williams
Science	Physics	Dysart	Williams
Language	Russian	Hall	Williams
Art	Sculpture	Bailey	Williams
History	World History	Clements	Williams
Math	Calculus	Steinbacner	Williams

## Example 10.20.2

Do the same selection as in the previous example but this time assign correlation names to the tables and use them to qualify the columns. Again we use IS\_IN here instead of ANY.

# UNCLASSIFIED

Just after the "use TYPES.ADA\_SQL;" statement insert the correlation redeclaration statements:

```
package D is new DEPARTMENT_CORRELATION.NAME ( "D" );
package C is new COURSE_CORRELATION.NAME ( "C" );
package P is new PROFESSOR_CORRELATION.NAME ( "P" );
package S is new STUDENT_CORRELATION.NAME ( "S" );
package CL is new CLASS_CORRELATION.NAME ( "CL" );
```

--Example 10.20.2

```
--      select D.DEPT_DESC, C.COURSE_DESC, P.PROF_NAME, S.ST_NAME
--      from DEPARTMENT D, COURSE C, PROFESSOR P, STUDENT S, CLASS CL
--      where S.ST_ID = any
--            ( select CLASS_STUDENT
--              from CLASS
--              group by CLASS_STUDENT
--              having COUNT (*) >= 4 )
--      and CL.CLASS_STUDENT = S.ST_ID
--      and CL.CLASS_COURSE = C.COURSE_ID
--      and C.COURSE_DEPT = D.DEPT_ID
--      and C.COURSE_PROF = P.PROF_ID ;

NEW_LINE;
PUT_LINE ("Output of Example 10.20.2");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( D.DEPT_DESC & C.COURSE_DESC & P.PROF_NAME & S.ST_NAME,
    FROM => D.DEPARTMENT & C.COURSE & P.PROFESSOR & S.STUDENT & CL.CLASS,
    WHERE => IS_IN ( S.ST_ID,
      SELEC ( CLASS_STUDENT,
        FROM => CLASS,
        GROUP_BY => CLASS_STUDENT,
        HAVING => COUNT ( '*' ) >= 4 ) )
    AND EQ ( CL.CLASS_STUDENT, S.ST_ID )
    AND EQ ( CL.CLASS_COURSE, C.COURSE_ID )
    AND EQ ( C.COURSE_DEPT, D.DEPT_ID )
    AND EQ ( C.COURSE_PROF, P.PROF_ID ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("DEPT_DESC  COURSE_DESC          PROF_NAME      ST_NAME");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    GOT_ONE := GOT_ONE + 1;
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    INTO ( V_COURSE_DESC, V_COURSE_DESC_INDEX);
```

# UNCLASSIFIED

```

    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);

    SET_COL (1);  -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    SET_COL (12); -- COURSE_DESC
    PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    SET_COL (34); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (48); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.20.2

DEPT_DESC	COURSE_DESC	PROF_NAME	ST_NAME
Language	French	Hall	Guiffre
Language	Russian	Hall	Guiffre
Art	Dance	Clements	Guiffre
Language	Spanish	Clements	Guiffre
History	World History	Clements	Williams
Science	Physics	Dysart	Williams
Language	Russian	Hall	Williams
Art	Sculpture	Bailey	Williams
History	World History	Clements	Williams
Math	Calculus	Steinbacner	Williams

## 10.21 Self Joins

There will be times when you wish to join the same table together as two or more tables. To do this you must use correlation names for the tables and then qualify the column names.

### Example 10.21.1

List the names and salaries of the professors earning the same amount or more than Professor Hall.

# UNCLASSIFIED

You will need to define two correlation names for the PROFESSOR table:

```
package X is new PROFESSOR_CORRELATION.NAME ( "X" );
package Y is new PROFESSOR_CORRELATION.NAME ( "Y" );
```

--Example 10.21.1

```
--      select X.PROF_NAME, X.PROF_SALARY
--      from PROFESSOR X, PROFESSOR Y
--      where X.PROF_SALARY >= Y.PROF_SALARY
--      and Y.PROF_NAME = 'Hall          ' ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.21.1");

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( X.PROF_NAME & X.PROF_SALARY,
    FROM => X.PROFESSOR & Y.PROFESSOR,
    WHERE => X.PROF_SALARY >= Y.PROF_SALARY
    AND EQ ( Y.PROF_NAME, "Hall          " ) ) ) ;
```

OPEN ( CURSOR );

begin

NEW\_LINE;

PUT\_LINE ("PROF\_NAME PROF\_SALARY");

GOT\_ONE := 0;

loop

FETCH ( CURSOR );

INTO ( V\_PROF\_NAME, V\_PROF\_NAME\_INDEX );

INTO ( V\_PROF\_SALARY );

GOT\_ONE := GOT\_ONE + 1;

SET\_COL (1); -- PROF\_NAME

PUT (V\_PROF\_NAME, V\_PROF\_NAME\_INDEX);

SET\_COL (15); -- PROF\_SALARY

PUT (V\_PROF\_SALARY, 5, 2, 0);

NEW\_LINE;

end loop;

exception

when NOT\_FOUND\_ERROR => if GOT\_ONE = 0 then

PUT\_LINE ("EXCEPTION: Not Found Error");

else

null;

end if;

when NO\_UPDATE\_ERROR => PUT\_LINE ("EXCEPTION: No Update Error");

when UNIQUE\_ERROR => PUT\_LINE ("EXCEPTION: Unique Error");

end;



# UNCLASSIFIED

```
CLOSE ( CURSOR );
```

## Output of Example 10.21.1

PROF_NAME	PROF_SALARY
Hall	45000.00
Bailey	50000.00

You only need to assign correlation names to tables where confusion might arise. You may select from several tables where only some of them have correlation names.

## Example 10.21.2

List the names, salaries, department and courses taught for the professors earning the same amount or more than Professor Hall.

Again you'll need two correlation names for the PROFESSOR table.

```
package X is new PROFESSOR_CORRELATION.NAME ( "X" );
package Y is new PROFESSOR_CORRELATION.NAME ( "Y" );
```

--Example 10.21.2

```
--      select X.PROF_FIRST, X.PROF_NAME, X.PROF_SALARY, DEPT_DESC, COURSE_DESC
--      from PROFESSOR X, PROFESSOR Y, DEPARTMENT, COURSE
--      where X.PROF_SALARY >= Y.PROF_SALARY
--            and Y.PROF_NAME = 'Hall'
--            and X.PROF_ID = COURSE_PROF
--            and COURSE_DEPT = DEPT_ID ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.21.2");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( X.PROF_FIRST & X.PROF_NAME & X.PROF_SALARY & DEPT_DESC &
    COURSE_DESC,
  FROM => X.PROFESSOR & Y.PROFESSOR & DEPARTMENT & COURSE,
  WHERE => X.PROF_SALARY >= Y.PROF_SALARY
    AND EQ ( Y.PROF_NAME, "Hall" )
    AND EQ ( X.PROF_ID, COURSE_PROF )
    AND EQ ( COURSE_DEPT, DEPT_ID ) ) ) ;
```

```
OPEN ( CURSOR );
```

```
begin
  NEW_LINE;
  PUT_LINE ("PROF_FIRST  PROF_NAME      PROF_SALARY  DEPT_DESC  COURSE_DESC");
  GOT_ONE := 0;
```

# UNCLASSIFIED

```

loop
  FETCH ( CURSOR );
  INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
  INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
  INTO ( V_PROF_SALARY );
  INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
  INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- PROF_FIRST
  PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
  SET_COL (13); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
  SET_COL (27); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
  SET_COL (40); -- DEPT_DESC
  PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
  SET_COL (51); -- COURSE_DESC
  PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.21.2

PROF_FIRST	PROF_NAME	PROF_SALARY	DEPT_DESC	COURSE_DESC
Elizabeth	Hall	45000.00	Language	French
Elizabeth	Hall	45000.00	Language	Russian
Bruce	Bailey	50000.00	Art	Sculpture
Bruce	Bailey	50000.00	Art	Music

## 10.22 EXISTS

EXISTS is a logical operator which is used in the WHERE clause before a subquery. It will return a true if the subquery returns at least one record and a false if it returns no records. The record being studied in the outer query is selected or rejected based on the true or false status of the subquery following the exists operator. The subquery associated with the EXISTS operator must refer to at least

UNCLASSIFIED

one column in the outer query if results are to be correct. If no column is referred to in the outer query then all queries will be selected or rejected based on the result of the subquery which will always be the same.

EXISTS has not been implemented in Level 1 of Ada/SQL. If you attempt to scan a program using an EXISTS operator in a query you will get the error message "%ADASQL-E-SCAN, Identifier has no valid meaning in this context". For the following examples I will show the query as it would be if EXISTS existed.

**Example 10.22.1**

List all professors who earn more than \$40000.00. Use the EXISTS operator to perform this query. This is not a good example of when the EXISTS operator should be used. It is to show the importance of referencing a column in the outer query.

You will need the correlation declaration:

```
package X is new PROFESSOR_CORRELATION.NAME ( "X" );

--Example 10.22.1

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => X.PROFESSOR,
    WHERE => EXISTS
      SELEC ( PROF_ID,
        FROM => PROFESSOR,
        WHERE => X.PROF_SALARY > 40000.00 ) ) ) ;
```

Note how we use a correlation name for the professor table in the outer query and then used that table for the prof\_salary column in the inner query. This query says look at every record in the professor table and for each record if the prof\_salary column is greater than \$40000.00 return true and select this record.

**Example 10.22.2**

Now do the same query without the correlation name.

```
--Example 10.22.2

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => PROFESSOR,
    WHERE => EXISTS
      SELEC ( PROF_ID,
        FROM => PROFESSOR,
        WHERE => PROF_SALARY > 40000.00 ) ) ) ;
```

In this case the subquery would return true if one or more record is selected. Without a reference to a

column in the outer query the result of the subquery will always be the same. Since in this case the subquery is true all records are selected for the outer query.

### Example 10.22.3

I want to know the average salary earned by professors teaching one or more courses with more than three semester hours. Be sure to count a professor's salary only one regardless of the number of qualifying courses he teaches.

--Example 10.22.3

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( avg (PROF_SALARY),
    FROM => PROFESSOR,
    WHERE => EXISTS
      SELEC ( COURSE_PROF,
        FROM => COURSE,
        WHERE => COURSE_HOURS > THREE
          AND EQ ( PROF_ID, COURSE_PROF ) ) ) ) ;

```

Note that I did not use a correlation name for the table in the outer query. It was not necessary in this case since any column from the table professor could only come from the outer query since the inner query is selecting only from the table course. This query must use the exists operator in order to get the correct results. We counted each professor only once regardless of how many courses of over three credit hours he teaches.

### Example 10.22.4

List the salary and professor id from all the records selected to form the average in the previous example.

--Example 10.22.4

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( PROF_SALARY & PROF_ID,
    FROM => PROFESSOR,
    WHERE => EXISTS
      SELEC ( COURSE_PROF,
        FROM => COURSE,
        WHERE => COURSE_HOURS > THREE
          AND EQ ( PROF_ID, COURSE_PROF ) ) ) ) ;

```

We would select three professors, no duplicates.

### Example 10.22.5

# UNCLASSIFIED

Now do the same query without the exists operator. Simply select the average salary for professors where the professor's id matches the teacher for a course from the course table and that course is more than three semester hours.

--Example 10.22.5

```
--      select avg (PROF_SALARY)
--      from PROFESSOR, COURSE
--      where COURSE_HOURS > 3
--      and PROF_ID = COURSE_PROF ;

NEW_LINE;
PUT_LINE ("Output of Example 10.22.5");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( avg (PROF_SALARY),
        FROM => PROFESSOR & COURSE,
        WHERE => COURSE_HOURS > THREE
        AND EQ ( PROF_ID, COURSE_PROF ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("AVG PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

Output of Example 10.22.5

UNCLASSIFIED

AVG PROF\_SALARY  
33571.43

We come up with a different average than the version with the EXISTS operator would have. This is because we counted the salary for each professor for each course they teach. Some of the professors must have been counted more than once to arrive at this average.

**Example 10.22.6**

List the records which were used to arrive at the average in the above example.

--Example 10.22.6

```
--      select PROF_SALARY, PROF_ID, COURSE_HOURS, COURSE_ID
--      from PROFESSOR, COURSE
--      where COURSE_HOURS > 3
--      and PROF_ID = COURSE_PROF ;

NEW_LINE;
PUT_LINE ("Output of Example 10.22.6");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_SALARY & PROF_ID & COURSE_HOURS & COURSE_ID,
        FROM => PROFESSOR & COURSE,
        WHERE => COURSE_HOURS > THREE
        AND EQ ( PROF_ID, COURSE_PROF ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_SALARY  PROF_ID  COURSE_HOURS  COURSE_ID");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_SALARY );
    INTO ( V_PROF_ID );
    INTO ( V_COURSE_HOURS );
    INTO ( V_COURSE_ID );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    SET_COL (14); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (23); -- COURSE_HOURS
    PUT (V_COURSE_HOURS);
    SET_COL (37); -- COURSE_ID
    PUT (V_COURSE_ID, 3);
```

## UNCLASSIFIED

```
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.22.6

PROF_SALARY	PROF_ID	COURSE_HOURS	COURSE_ID
35000.00	1	FIVE	302
35000.00	1	FOUR	303
45000.00	2	FOUR	403
30000.00	3	FOUR	201
30000.00	3	FIVE	203
30000.00	3	FOUR	204
30000.00	3	FOUR	202

As you can see professor one and three were selected multiple times since they teach several courses of over three semester hours.

### 10.23 INSERT INTO

The `INSERT INTO` statement is used to add new records to a table. The values added as columns of a record may be variables, literals or values returned as the result of a subquery. In section 10.2 we inserted literals into the columns of our tables. The format of the `INSERT INTO` statement is:

```
INSERT INTO ( table,
VALUES <= variable/literal/subquery_results AND
          variable/literal/subquery_results AND
          ... ) ;
```

You must supply data for every column in the table. Character strings must be enclosed in single quotes. Character string columns should be the maximum full length of the column. When a character string field won't fill up the column it is advisable that you pad it with spaces. The unused characters in a character string must be ascii spaces when using Ada/SQL. Otherwise you may end up with a data type incompatibility when accessing the field in an Ada/SQL program. Some DBMSs will automatically pad with spaces. Others will pad with a null value which is not ascii spaces. If you are not sure how your DBMS will pad a character string fill it with spaces yourself. Likewise with numeric fields, pick your own "null" value and stick to it. The null value assigned by the DBMS may not be compatible with Ada/SQL. In our examples we use zeros to pad numeric fields. All variables, literals and/or

## UNCLASSIFIED

subquery results must be of the same data type as the columns into which they are being inserted. It may be necessary to preform Ada type conversion.

### Example 10.23.1

Add a new record to the STUDENT table. This will be student number 26, Samuel Brenner, from California majoring in Art. This is his first year. We'll put him in dorm room A101.

--Example 10.23.1

```
--      insert into STUDENT
--      values
--      ( 026, 'Brenner      ', 'Samuel      ', 'A101', 'CA', 5, 1 ) ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.23.1");
```

```
INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(026) and
          TYPES.ADA_SQL.LAST_NAME'("Brenner      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Samuel      ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("A101") and
          TYPES.ADA_SQL.HOME_STATE'("CA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          ONE ) ;
```

### Output of Example 10.23.1

If you do not want to insert a value into each column of the record you may specify which columns you are supplying data for. All remaining columns will contain the null value designated by your DBMS. This may cause problems with data type compatibility in Ada/SQL. Before using this method be sure you know what your DBMS null values are. When specifying the columns to fill with data in an INSERT INTO statement the format is:

```
INSERT INTO ( table ( column & column & ... ) ,
VALUES <= variable/literal/subquery_results AND
          variable/literal/subquery_results AND
          ... ) ;
```

You do not have to enter the columns in the order in which they appear in the table. But you must enter the column names in the same order as the values to be inserted into the columns.

Remember, when inserting data into a column which is designated as "not\_null" you will get an error if the insert value is null. Also, when a column is designated as "unique" you will get an error if the data being inserted into that column is a duplicate of the data in that column of another record.

### Example 10.23.2



# UNCLASSIFIED

Add a student who we don't know much about. The only information is a last name of Mamout, from Alaska and this is the first year of study. Don't assign a student id number yet. We'll do that later on when we know more about this student.

```
INSERT INTO ( STUDENT ( ST_ID & ST_YEAR & ST_STATE & ST_NAME ),
VALUES <= ONE and
        TYPES.ADA_SQL.HOME_STATE'("AK") and
        TYPES.ADA_SQL.LAST_NAME'("Mamout      ") ) ;
```

The query above would be the one described by this example. However if we insert data into our table using this query the fields not filled end up with null values incompatible with our column data type. Therefore when we later attempt to retrieve data from those columns we get a constraint error. So I'm going to go ahead and fill those columns not used with my own null value.

--Example 10.23.2

```
--      insert into STUDENT
--      ( ST_YEAR, ST_STATE, ST_NAME )
--      values
--      ( 1, 'AK', 'Mamout      ' ) ;
--
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.23.2");
```

```
INSERT INTO ( STUDENT ( ST_ID & ST_YEAR & ST_STATE & ST_NAME & ST_FIRST &
        ST_MAJOR & ST_ROOM),
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(99) and
        ONE and
        TYPES.ADA_SQL.HOME_STATE'("AK") and
        TYPES.ADA_SQL.LAST_NAME'("Mamout      ") and
        TYPES.ADA_SQL.FIRST_NAME'("      ") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
        TYPES.ADA_SQL.GENERAL_ARRAY'("      ") ) ;
```

Output of Example 10.23.2

Example 10.23.3

Before inserting the last two students in our table we had students with id numbers between 1 and 25. Let's list all students who's id falls outside that range.

--Example 10.23.3

```
--      select *
--      from STUDENT
--      where ST_ID not between 1 and 25 ;
```

```
NEW_LINE;
```

# UNCLASSIFIED

```

PUT_LINE ("Output of Example 10.23.3");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => STUDENT,
    WHERE => NOT BETWEEN ( ST_ID, 1 and 25 ) ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");

```

# UNCLASSIFIED

end;

CLOSE ( CURSOR );

## Output of Example 10.23.3

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
26	Brenner	Samuel	A101	CA	5	ONE
99	Mamout			AK	1	ONE

## Example 10.23.4

Let's pull up Mamout's record by selecting all student who's name starts with an M. Normally we would want to compare ST\_NAME to "M%", however since ST\_NAME is a constrained array the compiler will not allow this. Therefore we must fill the comparison field to the full length of the ST\_NAME array.

--Example 10.23.4

```
--      select *
--      from STUDENT
--      where ST_NAME like ('M%');
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.23.4");

DECLAR ( CURSOR , CURSOR\_FOR =>

SELEC ( '\*',

FROM => STUDENT,

WHERE => LIKE ( ST\_NAME, ("M????????????") ) ) ) ;

OPEN ( CURSOR );

begin

NEW\_LINE;

PUT ("ST\_ID ST\_NAME ST\_FIRST ST\_ROOM ST\_STATE " &  
"ST\_MAJOR ST\_YEAR");

GOT\_ONE := 0;

loop

FETCH ( CURSOR );

INTO (V\_ST\_ID);

INTO (V\_ST\_NAME, V\_ST\_NAME\_INDEX);

INTO (V\_ST\_FIRST, V\_ST\_FIRST\_INDEX);

INTO (V\_ST\_ROOM, V\_ST\_ROOM\_INDEX);

INTO (V\_ST\_STATE, V\_ST\_STATE\_INDEX);

INTO (V\_ST\_MAJOR);

INTO (V\_ST\_YEAR);

GOT\_ONE := GOT\_ONE + 1;

# UNCLASSIFIED

```

SET_COL (1); -- ST_ID
PUT (V_ST_ID, 3);
SET_COL (6); -- ST_NAME
PUT (V_ST_NAME, V_ST_NAME_INDEX);
SET_COL (22); -- ST_FIRST
PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
SET_COL (34); -- ST_ROOM
PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
SET_COL (43); -- ST_STATE
PUT (V_ST_STATE, V_ST_STATE_INDEX);
SET_COL (53); -- ST_MAJOR
PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
PUT (V_ST_YEAR);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.23.4

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
2	McGinn	Gregory	A102	MD	1	THREE
5	McNamara	Howard	A201	VA	5	ONE
20	McMurray	Eric	B204	VA	2	ONE
22	Martin	Charlotte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE
99	Mamout			AK	1	ONE

Instead of supplying literals or values in variables for the data to be inserted into columns the results of a subquery can be used. All the records selected by the subquery will be inserted into the table. The subquery SELEC takes the place of the VALUES clause. If you don't list any column names all columns of the table will be filled. If you specify column names to be filled then only those columns will contain data. The format of the subquery INSERT INTO statement where all columns are to be filled is:

```

INSERT INTO ( table (column & column & ... ) ,
SELEC ( columns,
FROM => tables,

```

UNCLASSIFIED

remaining clauses in subquery ) ) ;

The columns listed in the SELEC clause must be in the same order and of compatible data types as the columns listed in the table in the INSERT INTO clause. If only specific columns are to be filled then those columns listed in the INSERT INTO clause will receive data from the columns listed in the SELEC clause, in the same order.

**Example 10.23.5**

Remember our GRADES table which we have left empty so far? Let's check it to see if it's still empty.

--Example 10.23.5

```
--      select *
--      from GRADE ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.23.5");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => GRADE) );
```

```
OPEN ( CURSOR );
```

```
begin
```

```
NEW_LINE;
PUT_LINE ("GRADE_COURSE GRADE_AVERAGE");
GOT_ONE := 0;
```

```
loop
```

```
  FETCH ( CURSOR );
  INTO (V_GRADE_COURSE);
  INTO (V_GRADE_AVERAGE);
  GOT_ONE := GOT_ONE + 1;
```

```
  SET_COL (1); -- GRADE_COURSE
  PUT (V_GRADE_COURSE, 3);
  SET_COL (14); -- GRADE_AVERAGE
  PUT (V_GRADE_AVERAGE, 3, 2, 0);
  NEW_LINE;
```

```
end loop;
```

```
exception
```

```
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                          PUT_LINE ("EXCEPTION: Not Found Error");
                          else
                            null;
                          end if;
```

```
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
```

## UNCLASSIFIED

```
when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

```
CLOSE ( CURSOR );
```

### Output of Example 10.23.5

GRADE\_COURSE GRADE\_AVERAGE

EXCEPTION: Not Found Error

### Example 10.23.6

Let's insert into the GRADE table all classes taught in department 5 and the average grade earned in those classes.

--Example 10.23.6

```
--      insert into GRADE
--      select CLASS_COURSE, avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--      from CLASS
--      where CLASS_DEPT = 5
--      group by CLASS_COURSE ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.23.6");
```

```
INSERT INTO ( GRADE ,
  SELEC ( CLASS_COURSE & avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) ,
  FROM => CLASS,
  WHERE => EQ ( CLASS_DEPT, 5 ) ,
  GROUP_BY => CLASS_COURSE ) ) ;
```

### Output of Example 10.23.6

### Example 10.23.7

List out the contents of the GRADE table.

--Example 10.23.7

```
--      select *
--      from GRADE ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.23.7");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
```

# UNCLASSIFIED

```

SELEC ( '*',
      FROM => GRADE) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("GRADE_COURSE GRADE_AVERAGE");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_GRADE_COURSE);
    INTO (V_GRADE_AVERAGE);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- GRADE_COURSE
    PUT (V_GRADE_COURSE, 3);
    SET_COL (14); -- GRADE_AVERAGE
    PUT (V_GRADE_AVERAGE, 3, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.23.7

GRADE_COURSE	GRADE_AVERAGE
501	82.86
502	68.53
503	89.00

## Example 10.23.8

Insert into the GRADE table the average grade for all classes. In the interactive query we left the course column of the table empty. We cannot do that here since future selection of the records will result in a constraint error. Therefore I will fill all columns of GRADE\_COURSE with a constant 999. Note that the 999 requires an Ada type conversion. The data to fill the column GRADE\_AVERAGE requires an Ada/SQL type conversion.

UNCLASSIFIED

```
--Example 10.23.8
--      insert into GRADE ( GRADE_AVERAGE )
--      select avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--      from CLASS ;

NEW_LINE;
PUT_LINE ( "Output of Example 10.23.8" );

INSERT INTO ( GRADE ( GRADE_COURSE & GRADE_AVERAGE ),
  SELEC ( TYPES.ADA_SQL.ID_COURSE'( 999 ) &
    CONVERT_TO.TYPES.GRADE_POINT
      ( avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) ),
  FROM => CLASS ) ) ;
```

Output of Example 10.23.8

Example 10.23.9

And list out the information in the GRADE table now.

```
--Example 10.23.9

--      select *
--      from GRADE ;

NEW_LINE;
PUT_LINE ( "Output of Example 10.23.9" );

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => GRADE ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ( "GRADE_COURSE GRADE_AVERAGE" );
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_GRADE_COURSE );
    INTO ( V_GRADE_AVERAGE );
    GOT_ONE := GOT_ONE + 1;

    SET_COL ( 1 ); -- GRADE_COURSE
    PUT ( V_GRADE_COURSE, 3 );
    SET_COL ( 14 ); -- GRADE_AVERAGE
    PUT ( V_GRADE_AVERAGE, 3, 2, 0 );
```



## UNCLASSIFIED

```
NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                           PUT_LINE ("EXCEPTION: Not Found Error");
                           else
                             null;
                           end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

### Output of Example 10.23.9

GRADE_COURSE	GRADE_AVERAGE
501	82.86
502	68.53
503	89.00
999	85.08

## 10.24 UPDATE

When you want to modify columns in a record in a table you will use the UPDATE statement. This allows you to change one or more columns in one or more records of a table. Columns may be modified with values in variables, literal values or expressions. A WHERE clause specifies the record(s) to be changed. If no WHERE clause is used, all records in the table will be changed. The format of the UPDATE statement is:

```
UPDATE ( table,
  SET => column <= value
  AND column <= value
  AND ... ,
  WHERE => where_expression ) ;
```

Value in the SET clause is either a variable value, a literal value or an expression. Expression in the WHERE clause is any expression valid in a WHERE clause and determines which records will be modified. We mentioned using the results of subqueries as the values to set columns in section 4. This is not allowed by standard SQL so Ada/SQL does not allow it either. If you attempt to scan an UPDATE query with a subquery used to set column values you will get the error "%ADASQL-E-SCAN, Identifier has no valid meaning in this context". Correlation names are not allowed in UPDATES either. Attempting to use a correlation name will result in the application scanner error "%ADASQL- E-SCAN, Table name is undefined".

### Example 10.24.1

# UNCLASSIFIED

Remember the student, Mamout, from Alaska that we had just added. Let's take a look at his record again.

--Example 10.24.1

```
--      select *
--      from STUDENT
--      where ST_NAME = 'Mamout      ' ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.24.1");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM => STUDENT,
        WHERE => EQ ( ST_NAME, "Mamout      ") ) ) ;
```

```
OPEN ( CURSOR );
```

```
begin
```

```
NEW_LINE;
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
     "ST_MAJOR  ST_YEAR");
GOT_ONE := 0;
```

```
loop
```

```
  FETCH ( CURSOR );
  INTO (V_ST_ID);
  INTO (V_ST_NAME, V_ST_NAME_INDEX);
  INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
  INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
  INTO (V_ST_STATE, V_ST_STATE_INDEX);
  INTO (V_ST_MAJOR);
  INTO (V_ST_YEAR);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- ST_ID
  PUT (V_ST_ID, 3);
  SET_COL (8); -- ST_NAME
  PUT (V_ST_NAME, V_ST_NAME_INDEX);
  SET_COL (22); -- ST_FIRST
  PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
  SET_COL (34); -- ST_ROOM
  PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
  SET_COL (43); -- ST_STATE
  PUT (V_ST_STATE, V_ST_STATE_INDEX);
  SET_COL (53); -- ST_MAJOR
  PUT (V_ST_MAJOR, 1);
  SET_COL (63); -- ST_YEAR
  PUT (V_ST_YEAR);
  NEW_LINE;
```

# UNCLASSIFIED

```

end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.24.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
99	Mamout			AK	1	ONE

## Example 10.24.2

We now want to fill in the empty and defaulted columns in his record. His id will be 27, his first name is Mark, room number B101 and his major is Science. Let's update his record with this information.

--Example 10.24.2

```

--      update STUDENT
--      set ST_ID = 27,
--          ST_FIRST = 'Mark      ',
--          ST_ROOM  = 'B101',
--          ST_MAJOR = 3
--      where ST_NAME = 'Mamout    ' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.24.2");

UPDATE ( STUDENT,
  SET => ST_ID <= 27
    and ST_FIRST <= "Mark      "
    and ST_ROOM  <= "B101"
    and ST_MAJOR <= 3,
  WHERE => EQ ( ST_NAME, "Mamout    " ) ) ;

```

## Output of Example 10.24.2

This update statement uses literals to update all the columns which we wish to modify.

# UNCLASSIFIED

## Example 10.24.3

Display Mark Mamout's record with the updates.

--Example 10.24.3

```
--      select *
--      from STUDENT
--      where ST_NAME = 'Mamout' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.24.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM => STUDENT,
        WHERE => EQ ( ST_NAME, "Mamout" ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
```

# UNCLASSIFIED

```

NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.24.3

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
27	Mamout	Mark	B101	AK	3	ONE

## Example 10.24.4

List all records in the professor table.

--Example 10.24.4

```

--      select *
--      from PROFESSOR ;

NEW_LINE;
PUT_LINE ("Output of Example 10.24.4");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => PROFESSOR ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
    "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );

```

# UNCLASSIFIED

```

    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.24.4

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

## Example 10.24.5

We will now give a 5% raise to all professors who have been with the school for more than 10 years.

--Example 10.24.5

```

--      update PROFESSOR
--      set PROF_SALARY = PROF_SALARY * 1.05

```

# UNCLASSIFIED

```
--      where PROF_YEARS > 10 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.24.5");

UPDATE ( PROFESSOR,
        SET => PROF_SALARY <= PROF_SALARY * 1.05,
        WHERE => PROF_YEARS > 10 ) ;
```

## Output of Example 10.24.5

This update is done with an expression where the new salary will be equal to the old salary multiplied by 1.05 which results in a 5% raise.

## Example 10.24.6

Now let's look at all records which we updated in the above query.

```
--Example 10.24.6

--      select *
--      from PROFESSOR
--      where PROF_YEARS > 10 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.24.6");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM => PROFESSOR,
        WHERE => PROF_YEARS > 10 ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
```

# UNCLASSIFIED

```

GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.24.6

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
4	Bailey	Bruce	5	15	52500.00

## Example 10.24.7

We want to adjust professor Steinbacner's salary to 5% more than the average of our professors who have been working here less than five years.

```

UPDATE ( PROFESSOR ,
SET => SELEC ( ( avg ( PROF_SALARY ) * 1.05 ),
FROM => PROFESSOR,
WHERE => PROF_YEARS < 5 )
WHERE => EQ ( PROF_NAME, "Steinbacner " ) ) ;

```

The above query would be the desired one. The SQL standard does not allow subqueries to set the columns of an update, therefore Ada/SQL must disallow it too. We could get around that rule in Ada/SQL by selecting the average salary into a variable in the first query and then using that variable \* 1.05 to set the salary in this query. For this example we will change the query to simply give Steinbacner a



UNCLASSIFIED

5% raise.

--Example 10.24.7

```
--      update PROFESSOR
--      set PROF_SALARY =
--        ( select ( avg ( PROF_SALARY ) * 1.05 )
--          from PROFESSOR
--            where PROF_YEARS < 5 )
--      where PROF_NAME = 'Steinbacner ' ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.24.7");

```
UPDATE ( PROFESSOR ,
SET => PROF_SALARY <= PROF_SALARY * 1.05,
WHERE => EQ ( PROF_NAME, "Steinbacner " ) ) ;
```

**Output of Example 10.24.7**

This update is done using a expression to determine the new contents of the modified columns.

**Example 10.24.8**

And display professor Steinbacner's updated record.

--Example 10.24.8

```
--      select *
--      from PROFESSOR
--      where PROF_NAME = 'Steinbacner ' ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.24.8");

```
DECLAR ( CURSOR , CURSOR_FOR =>
SELEC ( '*' ,
FROM => PROFESSOR,
WHERE => EQ ( PROF_NAME, "Steinbacner " ) ) ) ;
```

OPEN ( CURSOR );

begin

NEW\_LINE;

PUT\_LINE ("PROF\_ID PROF\_NAME PROF\_FIRST PROF\_DEPT " &  
"PROF\_YEARS PROF\_SALARY");

GOT\_ONE := 0;

loop

# UNCLASSIFIED

```

FETCH ( CURSOR );
INTO ( V_PROF_ID );
INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
INTO ( V_PROF_DEPT );
INTO ( V_PROF_YEARS );
INTO ( V_PROF_SALARY );
GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.24.8

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
3	Steinbacner	Moris	2	1	31500.00

## Example 10.24.9

Give all of our professors the suggested raise stored in the salary table based on the number of years they have been with us. This query needs correlation names as well as setting columns equal to values from a subquery. It is totally bogus for Ada/SQL. However if such a thing were allowed it would look like this:

```
UPDATE ( X.PROFESSOR,
```

# UNCLASSIFIED

```

SET => PROF_SALARY <=
( SELEC ( ( PROF_SALARY + ( PROF_SALARY * SAL_RAISE ) ),
FROM => PROFESSOR, SALARY,
WHERE => BETWEEN ( X.PROF_YEARS, SAL_YEAR and SAL_END )
AND EQ ( X.PROF_NAME, PROF_NAME ) ) ) ;

```

## Example 10.24.10

And list out the new information in the professor table.

--Example 10.24.10

```

--      select *
--      from PROFESSOR ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.24.10");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
SELEC ( '*' ,
FROM => PROFESSOR ) );

```

```

OPEN ( CURSOR );

```

begin

```

NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
"PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

```

loop

```

FETCH ( CURSOR );
INTO ( V_PROF_ID );
INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
INTO ( V_PROF_DEPT );
INTO ( V_PROF_YEARS );
INTO ( V_PROF_SALARY );
GOT_ONE := GOT_ONE + 1;

```

```

SET_COL (1); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
PUT (V_PROF_YEARS, 2);

```

UNCLASSIFIED

```
SET_COL (59); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

**Output of Example 10.24.10**

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	31500.00
4	Bailey	Bruce	5	15	52500.00
5	Clements	Carol	1	4	40000.00

**10.25 DELETE\_FROM**

The DELETE\_FROM statement is used to remove one or more records from a table. A WHERE clause specifies which record(s) are to be deleted. If the WHERE clause is omitted then all records in the table are deleted. The WHERE clause may include any expressions valid in a WHERE clause including subqueries. The format of the DELETE\_FROM statement is:

```
DELETE_FROM ( table,
WHERE => any valid where clause ) ;
```

**Example 10.25.1**

We will delete the record for the student named Bennett but first list that student's record.

--Example 10.25.1

```
--      select *
--      from STUDENT
--      where ST_NAME = 'Bennett' ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.25.1");
```

UNCLASSIFIED

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => STUDENT,
    WHERE => EQ ( ST_NAME, "Bennett      " ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;

  exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
      PUT_LINE ("EXCEPTION: Not Found Error");
    else
      null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

# UNCLASSIFIED

```
CLOSE ( CURSOR );
```

## Output of Example 10.25.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
11	Bennett	Nellie	A303	PA	4	THREE

## Example 10.25.2

Now delete Bennett's student record.

```
--Example 10.25.2
```

```
--      delete STUDENT
--      where ST_NAME =      'Bennett      ' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.2");

DELETE_FROM ( STUDENT,
WHERE => EQ ( ST_NAME, "Bennett      " ) ) ;
```

## Output of Example 10.25.2

## Example 10.25.3

We now want to delete the student record for Martin, list the record first.

```
--Example 10.25.3
```

```
--      select *
--      from STUDENT
--      where ST_NAME = 'Martin      ' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.3");

DECLAR ( CURSOR , CURSOR_FOR =>
SELEC ( '*',
FROM => STUDENT,
WHERE => EQ ( ST_NAME, "Martin      " ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
```

# UNCLASSIFIED

```

"ST_MAJOR ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.25.3

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
22	Martin	Charoltte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE

We have two Martins, Edward is the one we wish to delete.

# UNCLASSIFIED

## Example 10.25.4

Delete the record from the student table for Edward Martin.

--Example 10.25.4

```
--      delete STUDENT
--      where ST_NAME =      'Martin      '
--      and ST_FIRST =      'Edward      ' ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.25.4");

```
DELETE_FROM ( STUDENT ,
WHERE => EQ ( ST_NAME, "Martin      " )
AND EQ ( ST_FIRST, "Edward      " ) ) ;
```

## Output of Example 10.25.4

## Example 10.25.5

Now list all records remaining in the student table for Bennett and Martin.

--Example 10.25.5

```
--      select *
--      from STUDENT
--      where ST_NAME = 'Bennett      '
--      or ST_NAME = 'Martin      ' ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.25.5");

```
DECLAR ( CURSOR , CURSOR_FOR =>
SELEC ( '*',
FROM => STUDENT,
WHERE => EQ ( ST_NAME, "Bennett      " )
OR EQ ( ST_NAME, "Martin      " ) ) ) ;
```

OPEN ( CURSOR );

begin

NEW\_LINE;

PUT ("ST\_ID ST\_NAME ST\_FIRST ST\_ROOM ST\_STATE " &  
"ST\_MAJOR ST\_YEAR");

GOT\_ONE := 0;

loop

FETCH ( CURSOR );

INTO (V\_ST\_ID);



# UNCLASSIFIED

```

    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.25.5

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
22	Martin	Charoltte	C102	DC	1	TWO

Charoltte Martin is the only one left since Edward Martin's and Nellie Bennett's records were deleted.

## Example 10.25.6

We now want to delete from the student table and the class table all records for the student who has the

# UNCLASSIFIED

lowest average class grade. First list that student's id, the course and the average grade.

--Example 10.25.6

```
--      select CLASS_STUDENT, CLASS_COURSE, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS
--      where CLASS_STUDENT =
--            ( select CLASS_STUDENT
--              from CLASS
--              where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--                    ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--                      from CLASS ) ) ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.25.6");

DECLAR ( CURSOR , CURSOR\_FOR =>

SELEC ( CLASS\_STUDENT & CLASS\_COURSE &  
 ( ( CLASS\_SEM\_1 + CLASS\_SEM\_2 ) / 2.00 ) ,

FROM => CLASS,

WHERE => EQ ( CLASS\_STUDENT,

SELEC ( CLASS\_STUDENT,

FROM => CLASS,

WHERE => EQ ( ( CLASS\_SEM\_1 + CLASS\_SEM\_2 ) / 2.00,

SELEC ( min ( ( CLASS\_SEM\_1 + CLASS\_SEM\_2 ) / 2.00 ) ,

FROM => CLASS ) ) ) ) ;

OPEN ( CURSOR );

begin

NEW\_LINE;

PUT\_LINE ("CLASS\_STUDENT CLASS\_COURSE CLASS\_SEM\_1 + CLASS\_SEM\_2 / 2.00");

GOT\_ONE := 0;

loop

FETCH ( CURSOR );

INTO (V\_CLASS\_STUDENT);

INTO (V\_CLASS\_COURSE);

INTO (AVG\_SEM\_1);

GOT\_ONE := GOT\_ONE + 1;

SET\_COL (1); -- CLASS\_STUDENT

PUT (V\_CLASS\_STUDENT, 3);

SET\_COL (16); -- CLASS\_COURSE

PUT (V\_CLASS\_COURSE, 3);

SET\_COL (31); -- AVG\_SEM\_1

PUT (AVG\_SEM\_1, 3, 2, 0);

NEW\_LINE;

end loop;

exception

# UNCLASSIFIED

```

when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;

when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR     => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.25.6

CLASS_STUDENT	CLASS_COURSE	CLASS_SEM_1 + CLASS_SEM_2 / 2.00
8	502	63.30

## Example 10.25.7

Now select all the information in the student table about this person. Use a nested query, not a "where" clause based on information gathered from the previous query.

--Example 10.25.7

```

--      select *
--      from STUDENT
--      where ST_ID =
--            ( select CLASS_STUDENT
--              from CLASS
--              where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--                    ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--                      from CLASS ) ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.7");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => STUDENT,
    WHERE => EQ ( ST_ID,
    SELEC ( CLASS_STUDENT,
    FROM => CLASS,
    WHERE => EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
    SELEC ( min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS ) ) ) ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;

```

# UNCLASSIFIED

```

PUT ("ST_ID ST_NAME ST_FIRST ST_ROOM ST_STATE " &
     "ST_MAJOR ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.25.7

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
8	Hagan	Carl	A204	PA	5	FOUR

## UNCLASSIFIED

### Example 10.25.8

We will now delete this student from the student table. Structure the delete statement to delete the student with the lowest class average, do not use information gathered in previous queries.

--Example 10.25.8

```
--      delete STUDENT
--      where ST_ID =
--          ( select CLASS_STUDENT
--            from CLASS
--            where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--                  ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--                    from CLASS ) ) ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.25.8");

```
DELETE_FROM ( STUDENT,
  WHERE => EQ ( ST_ID,
    SELEC ( CLASS_STUDENT,
      FROM => CLASS,
        WHERE => EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
          SELEC ( min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
            FROM => CLASS ) ) ) ) ;
```

### Output of Example 10.25.8

### Example 10.25.9

Now delete information about this student from the class table. Structure the delete statement to delete the student class information with the lowest class average, do not use information gathered in previous queries.

--Example 10.25.9

```
--      delete CLASS
--      where CLASS_STUDENT =
--          ( select CLASS_STUDENT
--            from CLASS
--            where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--                  ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--                    from CLASS ) ) ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.25.9");

```
DELETE_FROM ( CLASS,
  WHERE => EQ ( CLASS_STUDENT,
```

# UNCLASSIFIED

```

SELEC ( CLASS_STUDENT,
FROM => CLASS,
WHERE => EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
SELEC ( min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
FROM => CLASS ) ) ) ;

```

## Output of Example 10.25.9

### Example 10.25.10

Now let's take a look at what's left in the student table.

--Example 10.25.10

```

--      select *
--      from STUDENT ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.25.10");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
SELEC ( '*',
FROM => STUDENT ) );

```

```

OPEN ( CURSOR );

```

begin

```

NEW_LINE;
PUT ("ST_ID ST_NAME ST_FIRST ST_ROOM ST_STATE " &
"ST_MAJOR ST_YEAR");
GOT_ONE := 0;

```

loop

```

FETCH ( CURSOR );
INTO (V_ST_ID);
INTO (V_ST_NAME, V_ST_NAME_INDEX);
INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
INTO (V_ST_STATE, V_ST_STATE_INDEX);
INTO (V_ST_MAJOR);
INTO (V_ST_YEAR);
GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- ST_ID
PUT (V_ST_ID, 3);
SET_COL (8); -- ST_NAME
PUT (V_ST_NAME, V_ST_NAME_INDEX);
SET_COL (22); -- ST_FIRST
PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
SET_COL (34); -- ST_ROOM

```

# UNCLASSIFIED

```

        PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
SET_COL (43); -- ST_STATE
        PUT (V_ST_STATE, V_ST_STATE_INDEX);
SET_COL (53); -- ST_MAJOR
        PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
        PUT (V_ST_YEAR);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                        PUT_LINE ("EXCEPTION: Not Found Error");
                        else
                            null;
                        end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

## Output of Example 10.25.10

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR
2	McGinn	Gregory	A102	MD	1	THREE
3	Lewis	Molly	A103	PA	4	TWO
4	Waxler	Dennis	A104	NC	2	TWO
5	McNamara	Howard	A201	VA	5	ONE
6	Hess	Fay	A202	DC	3	THREE
7	Guiffre	Jennifer	A203	MD	4	ONE
9	Bearman	Rose	A301	VA	2	ONE
10	Thompson	Paul	A302	NC	1	THREE
12	Schmidt	John	A304	SC	5	TWO
13	Gevarter	Susan	B101	NY	5	FOUR
14	Sherman	Donald	B102	VA	3	THREE
15	Gorham	Milton	B103	WV	2	TWO
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR
18	Ratliff	Ann	B202	NY	5	ONE
19	Phung	Kim	B203	SC	2	TWO
20	McMurray	Eric	B204	VA	2	ONE
21	O'Leary	Peggy	C101	PA	3	FOUR
22	Martin	Charoltte	C102	DC	1	TWO
23	O'Day	Hilda	C103	NC	4	ONE
25	Chateauneuf	Chelsea	C105	VA	1	THREE
26	Brenner	Samuel	A101	CA	5	ONE
27	Mamout	Mark	B101	AK	3	ONE

UNCLASSIFIED

**Example 10.25.11**

Delete all information in all tables now. Start by deleting the contents of the grade table.

--Example 10.25.11

```
--      delete GRADE ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.11");

begin
  DELETE_FROM ( GRADE ) ;
exception
  when NO_UPDATE_ERROR => PUT_LINE ("NO_UPDATE_ERROR: deleting grade");
end;
```

**Output of Example 10.25.11**

**Example 10.25.12**

List the contents of the grade table.

--Example 10.25.12

```
--      select *
--      from GRADE ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.12");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*' ,
    FROM => GRADE ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("GRADE_COURSE GRADE_AVERAGE");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_GRADE_COURSE);
    INTO (V_GRADE_AVERAGE);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- GRADE_COURSE
    PUT (V_GRADE_COURSE, 3);
```



UNCLASSIFIED

```
        SET_COL (14);  -- GRADE_AVERAGE
        PUT (V_GRADE_AVERAGE, 3, 2, 0);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

**Output of Example 10.25.12**

```
GRADE_COURSE GRADE_AVERAGE
EXCEPTION: Not Found Error
```

**Example 10.25.13**

Delete the contents of the department table.

--Example 10.25.13

```
--      delete DEPARTMENT;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.13");

begin
    DELETE_FROM ( DEPARTMENT ) ;
exception
    when NO_UPDATE_ERROR => PUT_LINE ("NO_UPDATE_ERROR: deleting department");
end;
```

**Output of Example 10.25.13**

**Example 10.25.14**

Delete the contents of the professor table.

--Example 10.25.14

UNCLASSIFIED

```
--      delete PROFESSOR;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.14");

begin
  DELETE_FROM ( PROFESSOR ) ;
exception
  when NO_UPDATE_ERROR => PUT_LINE ("NO_UPDATE_ERROR: deleting professor");
end;
```

**Output of Example 10.25.14**

**Example 10.25.15**

Delete the contents of the course table.

```
--Example 10.25.15

--      delete COURSE;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.15");

begin
  DELETE_FROM ( COURSE ) ;
exception
  when NO_UPDATE_ERROR => PUT_LINE ("NO_UPDATE_ERROR: deleting course");
end;
```

**Output of Example 10.25.15**

**Example 10.25.16**

Delete the contents of the student table.

```
--Example 10.25.16

--      delete STUDENT;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.16");

begin
  DELETE_FROM ( STUDENT ) ;
exception
  when NO_UPDATE_ERROR => PUT_LINE ("NO_UPDATE_ERROR: deleting student");
```

UNCLASSIFIED

end;

**Output of Example 10.25.16**

**Example 10.25.17**

Delete the contents of the class table.

--Example 10.25.17

```
--      delete CLASS;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.17");

begin
  DELETE_FROM ( CLASS ) ;
exception
  when NO_UPDATE_ERROR => PUT_LINE ("NO_UPDATE_ERROR: deleting class");
end;
```

**Output of Example 10.25.17**

**Example 10.25.18**

Delete the contents of the salary table.

--Example 10.25.18

```
--      delete SALARY;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.18");

begin
  DELETE_FROM ( SALARY ) ;
exception
  when NO_UPDATE_ERROR => PUT_LINE ("NO_UPDATE_ERROR: deleting salary");
end;
```

**Output of Example 10.25.18**

And now all of our tables are empty.

## 11. All The Pieces Of The Sample Program

This section lists out all the pieces of the sample program. The authorization package, the type declaration package, the table declaration package, the variables declaration package, the conversions, the complete sample programs with all the queries and the output from the sample program.

### 11.1 The Authorization Package - AUTH\_PACK.ADA

```
with SCHEMA_DEFINITION;
use SCHEMA_DEFINITION;

package AUTH_PACK is
    function UNITED_UNIV_AUTH is new AUTHORIZATION_IDENTIFIER;
end AUTH_PACK;
```

### 11.2 The Data Type Definition Package - TYPES.ADA

```
package TYPES is

    package ADA_SQL is

        type ID_DEPARTMENT is range 1 .. 9;
        type DESCRIPTION_DEPARTMENT is array (1 .. 8) of CHARACTER;
        type ID_PROFESSOR is range 1 .. 99;
        subtype ID_PROFESSOR_NOT_NULL_UNIQUE is ID_PROFESSOR;
        type NAME_COMPONENT is new CHARACTER;
        type LAST_NAME_INDEX is range 1 .. 12;
        type LAST_NAME is array (LAST_NAME_INDEX) of NAME_COMPONENT;
        type FIRST_NAME_INDEX is range 1 .. 10;
        type FIRST_NAME is array (FIRST_NAME_INDEX) of NAME_COMPONENT;
        type YEARS_EMPLOYED is range 1 .. 99;
        type YEARLY_INCOME is digits 7 range 0.0 .. 99999.99;
        type ID_COURSE is range 1 .. 999;
        subtype ID_COURSE_NOT_NULL is ID_COURSE;
        type DESCRIPTION_COURSE is array (INTEGER range <>) of CHARACTER;
        type ENUMERATION_NUMBERS is (ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX,
            SEVEN, EIGHT, NINE, TEN);
        subtype SEMESTER_HOURS is ENUMERATION_NUMBERS range ONE .. FIVE ;
        type ID_STUDENT is range 1 .. 999;
        type GENERAL_INDEX is range 1 .. 10;
        type GENERAL_COMPONENT is new CHARACTER;
        type GENERAL_ARRAY is array (GENERAL_INDEX range <>) of GENERAL_COMPONENT;
        subtype HOME_STATE is GENERAL_ARRAY (1..2);
        subtype YEARS_ATTENDED is ENUMERATION_NUMBERS range ONE .. FOUR;
        type GRADE_POINT is digits 5 range 0.0 .. 100.0;
        type SALARY_RAISE is digits 4 range 0.001 .. 0.500 ;
        type TOTAL_INCOME is digits 9 range 0.00 .. 9999999.00 ;
```

UNCLASSIFIED

end ADA\_SQL;

end TYPES;

### 11.3 The Table Definition Package - TABLES.ADA

```
with SCHEMA_DEFINITION, AUTH_PACK, TYPES;  
use SCHEMA_DEFINITION, AUTH_PACK, TYPES;
```

package TABLES is

use TYPES.ADA\_SQL;

package ADA\_SQL is

SCHEMA\_AUTHORIZATION : IDENTIFIER := UNITED\_UNIV\_AUTH;

type DEPARTMENT is

```
record  
  DEPT_ID      : ID_DEPARTMENT;  
  DEPT_DESC    : DESCRIPTION_DEPARTMENT;  
end record;
```

type PROFESSOR is

```
record  
  PROF_ID      : ID_PROFESSOR_NOT_NULL_UNIQUE;  
  PROF_NAME    : LAST_NAME;  
  PROF_FIRST   : FIRST_NAME;  
  PROF_DEPT    : ID_DEPARTMENT;  
  PROF_YEARS   : YEARS_EMPLOYED;  
  PROF_SALARY  : YEARLY_INCOME;  
end record;
```

type COURSE is

```
record  
  COURSE_ID    : ID_COURSE_NOT_NULL;  
  COURSE_DEPT  : ID_DEPARTMENT;  
  COURSE_DESC  : DESCRIPTION_COURSE (1..20);  
  COURSE_PROF  : ID_PROFESSOR;  
  COURSE_HOURS : SEMESTER_HOURS;  
end record;
```

type STUDENT is

```
record  
  ST_ID      : ID_STUDENT;  
  ST_NAME    : LAST_NAME;  
  ST_FIRST   : FIRST_NAME;  
  ST_ROOM    : GENERAL_ARRAY (1..4);  
  ST_STATE   : HOME_STATE;  
  ST_MAJOR   : ID_DEPARTMENT;
```

## UNCLASSIFIED

```
        ST_YEAR          : YEARS_ATTENDED;
    end record;

type CLASS is
    record
        CLASS_STUDENT    : ID_STUDENT;
        CLASS_DEPT       : ID_DEPARTMENT;
        CLASS_COURSE     : ID_COURSE;
        CLASS_SEM_1      : GRADE_POINT;
        CLASS_SEM_2      : GRADE_POINT;
        CLASS_GRADE      : GRADE_POINT;
    end record;

type GRADE is
    record
        GRADE_COURSE     : ID_COURSE;
        GRADE_AVERAGE   : GRADE_POINT;
    end record;

type SALARY is
    record
        SAL_YEAR        : YEARS_EMPLOYED;
        SAL_END         : YEARS_EMPLOYED;
        SAL_MIN         : YEARLY_INCOME;
        SAL_MAX         : YEARLY_INCOME;
        SAL_RAISE       : SALARY_RAISE;
    end record;

end ADA_SQL;

end TABLES;
```

### 11.4 The Variable Definition Package - VARIABLES.ADA

```
with TYPES, CURSOR_DEFINITION, DATABASE;
use CURSOR_DEFINITION;

package VARIABLES is

    use TYPES.ADA_SQL;

    CURSOR          : CURSOR_NAME;

    V_DEPT_ID       : ID_DEPARTMENT;
    V_DEPT_DESC     : DESCRIPTION_DEPARTMENT;
    V_DEPT_DESC_INDEX : INTEGER;

    V_PROF_ID       : ID_PROFESSOR;
    V_PROF_NAME     : LAST_NAME;
```

# UNCLASSIFIED

```

V_PROF_NAME_INDEX      : LAST_NAME_INDEX;
V_PROF_FIRST           : FIRST_NAME;
V_PROF_FIRST_INDEX     : FIRST_NAME_INDEX;
V_PROF_DEPT            : ID_DEPARTMENT;
V_PROF_YEARS           : YEARS_EMPLOYED;
V_PROF_SALARY          : YEARLY_INCOME;

V_COURSE_ID            : ID_COURSE;
V_COURSE_DEPT          : ID_DEPARTMENT;
V_COURSE_DESC          : DESCRIPTION_COURSE (1..20);
V_COURSE_DESC_INDEX    : INTEGER;
V_COURSE_PROF          : ID_PROFESSOR;
V_COURSE_HOURS         : SEMESTER_HOURS;

V_ST_ID                : ID_STUDENT;
V_ST_NAME              : LAST_NAME;
V_ST_NAME_INDEX        : LAST_NAME_INDEX;
V_ST_FIRST             : FIRST_NAME;
V_ST_FIRST_INDEX       : FIRST_NAME_INDEX;
V_ST_ROOM              : GENERAL_ARRAY (1..4);
V_ST_ROOM_INDEX        : GENERAL_INDEX;
V_ST_STATE             : HOME_STATE;
V_ST_STATE_INDEX       : GENERAL_INDEX;
V_ST_MAJOR             : ID_DEPARTMENT;
V_ST_YEAR              : YEARS_ATTENDED;

V_CLASS_STUDENT        : ID_STUDENT;
V_CLASS_DEPT           : ID_DEPARTMENT;
V_CLASS_COURSE         : ID_COURSE;
V_CLASS_SEM_1          : GRADE_POINT;
V_CLASS_SEM_2          : GRADE_POINT;
V_CLASS_GRADE          : GRADE_POINT;

V_GRADE_COURSE         : ID_COURSE;
V_GRADE_AVERAGE       : GRADE_POINT;

V_SAL_YEAR             : YEARS_EMPLOYED;
V_SAL_END              : YEARS_EMPLOYED;
V_SAL_MIN              : YEARLY_INCOME;
V_SAL_MAX              : YEARLY_INCOME;
V_SAL_RAISE            : SALARY_RAISE;

COUNT_RESULT          : DATABASE.INT;
AVG_SALARY             : YEARLY_INCOME;
MIN_SALARY             : YEARLY_INCOME;
MAX_SALARY             : YEARLY_INCOME;
SUM_SALARY             : TOTAL_INCOME;
AVG_SEM_1              : GRADE_POINT;
AVG_SEM_2              : GRADE_POINT;

```

end VARIABLES;

## 11.5 The Conversion Package - CONVERSIONS.ADA

```

with TYPES, TEXT_IO;
  use TEXT_IO;

package CONVERSION_SUBS is

  use TYPES.ADA_SQL;

  -- each different type of component for arrays needs a routine to convert
  -- the individual components to CHARACTER components of a STRING

  function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in NAME_COMPONENT)
    return CHARACTER;

  function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in GENERAL_COMPONENT)
    return CHARACTER;

  function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in CHARACTER)
    return CHARACTER;

  -- we need a generic routine for the conversion of all constrained arrays
  -- to STRINGS

  generic
    type INDEX_TYPE is range <>;
    type COMPONENT_TYPE is (<>);
    type ARRAY_TYPE is array ( INDEX_TYPE ) of COMPONENT_TYPE;
    with function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
      ( CHAR_IN : COMPONENT_TYPE )
      return CHARACTER is <>;
  package CONSTRAINED_ARRAYS_IO is

    -- we call the routine PUT so in the DML program it's transparent that all
    -- kinds of conversions are taking place

    procedure PUT
      (STRING_IN : in ARRAY_TYPE;
       INDEX_IN : in INDEX_TYPE);

  end CONSTRAINED_ARRAYS_IO;

  -- we need a generic routine for the conversion of all unconstrained arrays
  -- to STRINGS

  generic
    type INDEX_TYPE is range <>;

```



UNCLASSIFIED

```

type COMPONENT_TYPE is (<>);
type ARRAY_TYPE is array ( INDEX_TYPE range <> ) of COMPONENT_TYPE;
with function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    ( CHAR_IN : COMPONENT_TYPE )
    return CHARACTER is <>;
package UNCONSTRAINED_ARRAYS_IO is

-- we call the routine PUT so in the DML program it's transparent that all
-- kinds of conversions are taking place

procedure PUT
    (STRING_IN  : in ARRAY_TYPE;
     INDEX_IN   : in INDEX_TYPE);
end UNCONSTRAINED_ARRAYS_IO;

end CONVERSION_SUBS;

package body CONVERSION_SUBS is

-- each different type of component for arrays needs a routine to convert
-- the individual components to CHARACTER components of a STRING

function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in NAME_COMPONENT)
    return CHARACTER is
begin
    return CHARACTER (CHAR_IN);
end CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS;

function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in GENERAL_COMPONENT)
    return CHARACTER is
begin
    return CHARACTER (CHAR_IN);
end CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS;

function CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
    (CHAR_IN : in CHARACTER)
    return CHARACTER is
begin
    return CHARACTER (CHAR_IN);
end CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS;

-- we need a generic routine for the conversion of all constrained arrays
-- to STRINGS

package body CONSTRAINED_ARRAYS_IO is

-- we call the routine PUT so in the DML program it's transparent that all
-- kinds of conversions are taking place

```

# UNCLASSIFIED

```

procedure PUT
  (STRING_IN  : in ARRAY_TYPE;
   INDEX_IN   : in INDEX_TYPE) is

  STRING_OUT : STRING (1..100);
  INDEX_OUT  : INTEGER;

begin
  INDEX_OUT := INTEGER (INDEX_IN);
  for I in 1.. INTEGER (INDEX_IN)
  loop
    STRING_OUT (I) := CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
                      (STRING_IN (INDEX_TYPE (I))) ;
  end loop;
  PUT (STRING_OUT (1..INDEX_OUT));
end PUT;

end CONSTRAINED_ARRAYS_IO;

-- we need a generic routine for the conversion of all unconstrained arrays
-- to STRINGS

package body UNCONSTRAINED_ARRAYS_IO is

-- we call the routine PUT so in the DML program it's transparent that all
-- kinds of conversions are taking place

procedure PUT
  (STRING_IN  : in ARRAY_TYPE;
   INDEX_IN   : in INDEX_TYPE) is

  STRING_OUT : STRING (1..100);
  INDEX_OUT  : INTEGER;

begin
  INDEX_OUT := INTEGER (INDEX_IN);
  for I in 1.. INTEGER (INDEX_IN)
  loop
    STRING_OUT (I) := CONVERT_ARRAY_COMPONENTS_TO_CHARACTERS
                      (STRING_IN (INDEX_TYPE (I))) ;
  end loop;
  PUT (STRING_OUT (1..INDEX_OUT));
end PUT;

end UNCONSTRAINED_ARRAYS_IO;

end CONVERSION_SUBS;

-- the package CONVERSIONS instantiates all the necessary packages so our
-- DML unit can "with" and "use" them and not be cluttered with all this

```

# UNCLASSIFIED

```
-- instantiation

with TEXT_IO, TYPES, CONVERSION_SUBS, DATABASE;
use TEXT_IO, CONVERSION_SUBS;

package CONVERSIONS is

    use TYPES.ADA_SQL;

-- to output character strings

    package CONVERT_LAST_NAME is new CONSTRAINED_ARRAYS_IO
        (LAST_NAME_INDEX, NAME_COMPONENT, LAST_NAME);

    package CONVERT_FIRST_NAME is new CONSTRAINED_ARRAYS_IO
        (FIRST_NAME_INDEX, NAME_COMPONENT, FIRST_NAME);

    package CONVERT_DESCRIPTION_COURSE is new UNCONSTRAINED_ARRAYS_IO
        (INTEGER, CHARACTER, DESCRIPTION_COURSE);

    package CONVERT_GENERAL_ARRAY is new UNCONSTRAINED_ARRAYS_IO
        (GENERAL_INDEX, GENERAL_COMPONENT, GENERAL_ARRAY);

-- to output integer data as strings

    package I1_CONVERT is new INTEGER_IO (ID_DEPARTMENT);
    package I2_CONVERT is new INTEGER_IO (ID_PROFESSOR);
    package I3_CONVERT is new INTEGER_IO (YEARS_EMPLOYED);
    package I4_CONVERT is new INTEGER_IO (ID_COURSE);
    package I5_CONVERT is new INTEGER_IO (ID_STUDENT);
    package I6_CONVERT is new INTEGER_IO (DATABASE.INT);

-- to output floating point data as strings

    package CONVERT_FLOAT_YEARLY_INCOME is new FLOAT_IO (YEARLY_INCOME);
    package CONVERT_FLOAT_GRADE_POINT is new FLOAT_IO (GRADE_POINT);
    package CONVERT_FLOAT_SALARY_RAISE is new FLOAT_IO (SALARY_RAISE);
    package CONVERT_FLOAT_TOTAL_INCOME is new FLOAT_IO (TOTAL_INCOME);

-- to output enumeration data as strings

    package CONVERT_ENUMERATION_ENUMERATION_NUMBERS is new
        ENUMERATION_IO (ENUMERATION_NUMBERS);

-- to output stuff from text_io since we can't "use" text_io cause COUNT is
-- redundant as in select count (*)

    procedure PUT_LINE (ITEM : in STRING ) renames TEXT_IO.PUT_LINE;
    procedure NEW_LINE (SPACING : in TEXT_IO.POSITIVE_COUNT := 1)
        renames TEXT_IO.NEW_LINE;
    procedure SET_COL (TO : in TEXT_IO.POSITIVE_COUNT) renames TEXT_IO.SET_COL;
```

## UNCLASSIFIED

```
procedure PUT (ITEM : in STRING ) renames TEXT_IO.PUT;

end CONVERSIONS;
```

### 11.6 The Sample Program - EXAMPLES.ADA

```
with TYPES, TABLES, VARIABLES;
  use TYPES, TABLES, VARIABLES;
with EXAMPLES_ADA_SQL;
  use EXAMPLES_ADA_SQL;
with TEXT_IO, CONVERSIONS;
  use CONVERSIONS;

procedure EXAMPLES is

  use TYPES.ADA_SQL;

  package D is new DEPARTMENT_CORRELATION.NAME ( "D" );
  package C is new COURSE_CORRELATION.NAME ( "C" );
  package P is new PROFESSOR_CORRELATION.NAME ( "P" );
  package S is new STUDENT_CORRELATION.NAME ( "S" );
  package CL is new CLASS_CORRELATION.NAME ( "CL" );

  package X is new PROFESSOR_CORRELATION.NAME ( "X" );
  package Y is new PROFESSOR_CORRELATION.NAME ( "Y" );

  -- to do all the data conversions for displaying information

  use CONVERT_LAST_NAME, CONVERT_FIRST_NAME, CONVERT_DESCRIPTION_COURSE,
    CONVERT_GENERAL_ARRAY, I1_CONVERT, I2_CONVERT, I3_CONVERT, I4_CONVERT,
    I5_CONVERT, I6_CONVERT, CONVERT_FLOAT_YEARLY_INCOME,
    CONVERT_FLOAT_TOTAL_INCOME, CONVERT_FLOAT_GRADE_POINT,
    CONVERT_FLOAT_SALARY_RAISE, CONVERT_ENUMERATION_ENUMERATION_NUMBERS;

  GOT_ONE : NATURAL := 0;

begin

  OPEN_DATABASE ( "SYSTEM", "MANAGER" );

  -- Example 10.1.1.1

  --      select *
  --      from DEPARTMENT ;

  NEW_LINE;
  PUT_LINE ( "Output of Example 10.1.1.1" );

  DECLAR ( CURSOR , CURSOR_FOR =>
```

UNCLASSIFIED

```

        SELEC ( '*',
              FROM => DEPARTMENT ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("DEPT_ID   DEPT_DESC");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO ( V_DEPT_ID );
        INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- DEPT_ID
        PUT (V_DEPT_ID, 1);
        SET_COL (11); -- DEPT_DESC
        PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.1.1.2

--      select *
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.1.1.2");

begin

    SELEC ( '*',
          FROM => DEPARTMENT );
        INTO ( V_DEPT_ID );
        INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );

    NEW_LINE;

```

# UNCLASSIFIED

```

PUT_LINE ("DEPT_ID  DEPT_DESC");
SET_COL (1);  -- DEPT_ID
PUT (V_DEPT_ID, 1);
SET_COL (11);  -- DEPT_DESC
PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
NEW_LINE;

exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

-- Example 10.1.2.1

--      select DEPT_DESC
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.1.2.1");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( DEPT_DESC,
    FROM => DEPARTMENT ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("          DEPT_DESC");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (11);  -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

# UNCLASSIFIED

```

CLOSE ( CURSOR );

-- Example 10.1.2.2

--      select DEPT_DESC
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.1.2.2");

begin

  SELEC ( DEPT_DESC,
    FROM => DEPARTMENT );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );

  NEW_LINE;
  PUT_LINE ( "          DEPT_DESC" );
  SET_COL (11); -- DEPT_DESC
  PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
  NEW_LINE;

exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error ");
end;

-- Example 10.2.1

NEW_LINE;
PUT_LINE ("Output of Example 10.2.1");

INSERT INTO ( DEPARTMENT ,
VALUES <= TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          TYPES.ADA_SQL.DESCRPTION_DEPARTMENT'("History ") ) ;

-- Example 10 2 2

--      select *
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.2.2");

begin
  SELEC ( '*',
    FROM => DEPARTMENT );
    INTO ( V_DEPT_ID );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );

```

# UNCLASSIFIED

```

NEW_LINE;
PUT_LINE ("DEPT_ID  DEPT_DESC");
SET_COL (1); -- DEPT_ID
PUT (V_DEPT_ID, 1);
SET_COL (11); -- DEPT_DESC
PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
NEW_LINE;

exception
when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

-- Example 10.2.3

--      select DEPT_DESC
--      from DEPARTMENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.2.3");

begin
SELEC ( DEPT_DESC,
FROM => DEPARTMENT );
INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );

NEW_LINE;
PUT_LINE ( "      DEPT_DESC");
SET_COL (11); -- DEPT_DESC
PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
NEW_LINE;

exception
when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

-- Example 10.2.4

NEW_LINE;
PUT_LINE ("Output of Example 10.2.4");

INSERT INTO ( DEPARTMENT ,
VALUES <= TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.DESCRPTION_DEPARTMENT'("Math      ") ) :

-- Example 10.2.5

```



# UNCLASSIFIED

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.5");

INSERT INTO ( DEPARTMENT ,
VALUES <= TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.DESCRPTION_DEPARTMENT'("Science ") ) ;
```

-- Example 10.2.6

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.6");

INSERT INTO ( DEPARTMENT ,
VALUES <= TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.DESCRPTION_DEPARTMENT'("Language") ) ;
```

-- Example 10.2.7

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.7");

INSERT INTO ( DEPARTMENT ,
VALUES <= TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.DESCRPTION_DEPARTMENT'("Art      ") ) ;
```

-- Example 10.2.8

```
--      select *
--      from DEPARTMENT ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.8");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*' ,
        FROM => DEPARTMENT ) );
```

```
OPEN ( CURSOR );
```

begin

```
NEW_LINE;
PUT_LINE ("DEPT_ID   DEPT_DESC");
GOT_ONE := 0;
```

loop

```
  FETCH ( CURSOR );
  INTO ( V_DEPT_ID );
  INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
  GOT_ONE := GOT_ONE + 1;
```

# UNCLASSIFIED

```

SET_COL (1); -- DEPT_ID
PUT (V_DEPT_ID, 1);
SET_COL (11); -- DEPT_DESC
PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

CLOSE ( CURSOR );

-- Example 10.2.9

```

NEW_LINE;
PUT_LINE ("Output of Example 10.2.9");

INSERT INTO ( PROFESSOR ,
VALUES <= TYPES.ADA_SQL.ID_PROFESSOR'(01) and
        TYPES.ADA_SQL.LAST_NAME'("Dysart      ") and
        TYPES.ADA_SQL.FIRST_NAME'("Gregory   ") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
        TYPES.ADA_SQL.YEARS_EMPLOYED'(03) and
        TYPES.ADA_SQL.YEARLY_INCOME'(35000.00) ) ;

```

-- Example 10.2.10

```

NEW_LINE;
PUT_LINE ("Output of Example 10.2.10");

INSERT INTO ( PROFESSOR ,
VALUES <= TYPES.ADA_SQL.ID_PROFESSOR'(02) and
        TYPES.ADA_SQL.LAST_NAME'("Hall      ") and
        TYPES.ADA_SQL.FIRST_NAME'("Elizabeth ") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
        TYPES.ADA_SQL.YEARS_EMPLOYED'(07) and
        TYPES.ADA_SQL.YEARLY_INCOME'(45000.00) ) ;

```

-- Example 10.2.11

```

NEW_LINE;
PUT_LINE ("Output of Example 10.2.11");

INSERT INTO ( PROFESSOR ,

```

# UNCLASSIFIED

```

VALUES <= TYPES.ADA_SQL.ID_PROFESSOR'(03) and
        TYPES.ADA_SQL.LAST_NAME('Steinbacner ") and
        TYPES.ADA_SQL.FIRST_NAME('Moris      ") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
        TYPES.ADA_SQL.YEARS_EMPLOYED'(01) and
        TYPES.ADA_SQL.YEARLY_INCOME'(30000.00) ) ;

-- Example 10.2.12

NEW_LINE;
PUT_LINE ("Output of Example 10.2.12");

INSERT INTO ( PROFESSOR ,
VALUES <= TYPES.ADA_SQL.ID_PROFESSOR'(04) and
        TYPES.ADA_SQL.LAST_NAME('Bailey      ") and
        TYPES.ADA_SQL.FIRST_NAME('Bruce      ") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
        TYPES.ADA_SQL.YEARS_EMPLOYED'(15) and
        TYPES.ADA_SQL.YEARLY_INCOME'(50000.00) ) ;

-- Example 10.2.13

NEW_LINE;
PUT_LINE ("Output of Example 10.2.13");

INSERT INTO ( PROFESSOR ,
VALUES <= TYPES.ADA_SQL.ID_PROFESSOR'(05) and
        TYPES.ADA_SQL.LAST_NAME('Clements    ") and
        TYPES.ADA_SQL.FIRST_NAME('Carol      ") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
        TYPES.ADA_SQL.YEARS_EMPLOYED'(04) and
        TYPES.ADA_SQL.YEARLY_INCOME'(40000.00) ) ;

-- Example 10.2.14

--      select *
--      from PROFESSOR ;

NEW_LINE;
PUT_LINE ("Output of Example 10.2.14");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM => PROFESSOR ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");

```

UNCLASSIFIED

GOT\_ONE := 0;

loop

```

    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

```

```

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);

```

NEW\_LINE;

end loop;

exception

```

    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

```

when NO\_UPDATE\_ERROR => PUT\_LINE ("EXCEPTION: No Update Error");

when UNIQUE\_ERROR => PUT\_LINE ("EXCEPTION: Unique Error");

end;

CLOSE ( CURSOR );

-- Example 10.2.15

NEW\_LINE;

PUT\_LINE ("Output of Example 10.2.15");

```

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(101) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
        TYPES.ADA_SQL.DESCRPTION_COURSE'("World History" and
        TYPES.ADA_SQL.ID_PROFESSOR'(05) and
        TWO ) ;

```

UNCLASSIFIED

-- Example 10.2.16

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.16");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(102) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Political History ") and
          TYPES.ADA_SQL.ID_PROFESSOR'(05) and
          THREE ) ;
```

-- Example 10.2.17

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.17");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(103) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Ancient History ") and
          TYPES.ADA_SQL.ID_PROFESSOR'(05) and
          TWO ) ;
```

-- Example 10.2.18

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.18");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(201) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Algebra ") and
          TYPES.ADA_SQL.ID_PROFESSOR'(03) and
          FOUR ) ;
```

-- Example 10.2.19

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.19");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(202) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Geometry ") and
          TYPES.ADA_SQL.ID_PROFESSOR'(03) and
          FOUR ) ;
```

-- Example 10.2.20

```
NEW_LINE;
```

# UNCLASSIFIED

```
PUT_LINE ("Output of Example 10.2.20");
```

```
INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(203) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Trigonometry" ) and
          TYPES.ADA_SQL.ID_PROFESSOR'(03) and
          FIVE ) ;
```

```
-- Example 10.2.21
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.21");
```

```
INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(204) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Calculus" ) and
          TYPES.ADA_SQL.ID_PROFESSOR'(03) and
          FOUR ) ;
```

```
-- Example 10.2.22
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.22");
```

```
INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(301) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Chemistry" ) and
          TYPES.ADA_SQL.ID_PROFESSOR'(01) and
          THREE ) ;
```

```
-- Example 10.2.23
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.23");
```

```
INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(302) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Physics" ) and
          TYPES.ADA_SQL.ID_PROFESSOR'(01) and
          FIVE ) ;
```

```
-- Example 10.2.24
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.24");
```

```
INSERT INTO ( COURSE ,
```

UNCLASSIFIED

```
VALUES <= TYPES.ADA_SQL.ID_COURSE'(303) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
        TYPES.ADA_SQL.DESCRPTION_COURSE'("Biology") and
        TYPES.ADA_SQL.ID_PROFESSOR'(01) and
        FOUR ) ;
```

-- Example 10.2.25

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.25");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(401) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
        TYPES.ADA_SQL.DESCRPTION_COURSE'("French") and
        TYPES.ADA_SQL.ID_PROFESSOR'(02) and
        TWO ) ;
```

-- Example 10.2.26

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.26");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(402) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
        TYPES.ADA_SQL.DESCRPTION_COURSE'("Spanish") and
        TYPES.ADA_SQL.ID_PROFESSOR'(05) and
        TWO ) ;
```

-- Example 10.2.27

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.27");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(403) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
        TYPES.ADA_SQL.DESCRPTION_COURSE'("Russian") and
        TYPES.ADA_SQL.ID_PROFESSOR'(02) and
        FOUR ) ;
```

-- Example 10.2.28

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.28");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(501) and
        TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
        TYPES.ADA_SQL.DESCRPTION_COURSE'("Sculpture") and
```

UNCLASSIFIED

```
TYPES.ADA_SQL.ID_PROFESSOR'(04) and
ONE ) ;
```

-- Example 10.2.29

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.29");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(502) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Music" and
          TYPES.ADA_SQL.ID_PROFESSOR'(04) and
          ONE ) ;
```

-- Example 10.2.30

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.30");

INSERT INTO ( COURSE ,
VALUES <= TYPES.ADA_SQL.ID_COURSE'(503) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.DESCRPTION_COURSE'("Dance" and
          TYPES.ADA_SQL.ID_PROFESSOR'(05) and
          TWO ) ;
```

-- Example 10.2.31

```
--      select *
--      from COURSE ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.31");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*' ,
        FROM => COURSE ) );
```

```
OPEN ( CURSOR );
```

begin

```
NEW_LINE;
PUT_LINE ("COURSE_ID  COURSE_DEPT COURSE_DESC" &
        "COURSE_PROF COURSE_HOURS");
GOT_ONE := 0;
```

loop

```
    FETCH ( CURSOR );
    INTO (V_COURSE_ID);
    INTO (V_COURSE_DEPT);
```



# UNCLASSIFIED

```

    INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    INTO (V_COURSE_PROF);
    INTO (V_COURSE_HOURS);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- COURSE_ID
    PUT (V_COURSE_ID, 3);
    SET_COL (12); -- COURSE_DEPT
    PUT (V_COURSE_DEPT, 1);
    SET_COL (24); -- COURSE_DESC
    PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    SET_COL (46); -- COURSE_PROF
    PUT (V_COURSE_PROF, 2);
    SET_COL (58); -- COURSE_HOURS
    PUT (V_COURSE_HOURS);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR     => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.2.32

NEW_LINE;
PUT_LINE ("Output of Example 10.2.32");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(001) and
        TYPES.ADA_SQL.LAST_NAME'("Horrigan ") and
        TYPES.ADA_SQL.FIRST_NAME'("William ") and
        TYPES.ADA_SQL.GENERAL_ARRAY'("A101") and
        TYPES.ADA_SQL.HOME_STATE'("VA") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
        FOUR ) ;

-- Example 10.2.33

NEW_LINE;
PUT_LINE ("Output of Example 10.2.33");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(002) and

```

# UNCLASSIFIED

```
TYPES.ADA_SQL.LAST_NAME'("McGinn      ") and
TYPES.ADA_SQL.FIRST_NAME'("Gregory   ") and
TYPES.ADA_SQL.GENERAL_ARRAY'("A102") and
TYPES.ADA_SQL.HOME_STATE'("MD") and
TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
THREE ) ;
```

-- Example 10.2.34

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.34");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(003) and
          TYPES.ADA_SQL.LAST_NAME'("Lewis      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Molly     ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("A103") and
          TYPES.ADA_SQL.HOME_STATE'("PA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TWO ) ;
```

-- Example 10.2.35

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.35");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(004) and
          TYPES.ADA_SQL.LAST_NAME'("Waxler     ") and
          TYPES.ADA_SQL.FIRST_NAME'("Dennis    ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("A104") and
          TYPES.ADA_SQL.HOME_STATE'("NC") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TWO ) ;
```

-- Example 10.2.36

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.36");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(005) and
          TYPES.ADA_SQL.LAST_NAME'("McNamara    ") and
          TYPES.ADA_SQL.FIRST_NAME'("Howard     ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("A201") and
          TYPES.ADA_SQL.HOME_STATE'("VA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          ONE ) ;
```

-- Example 10.2.37

UNCLASSIFIED

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.37");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(006) and
          TYPES.ADA_SQL.LAST_NAME'("Hess      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Fay      ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A202") and
          TYPES.ADA_SQL.HOME_STATE'("DC") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          THREE ) ;
```

-- Example 10.2.38

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.38");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(007) and
          TYPES.ADA_SQL.LAST_NAME'("Guiffre   ") and
          TYPES.ADA_SQL.FIRST_NAME'("Jennifer ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A203") and
          TYPES.ADA_SQL.HOME_STATE'("MD") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          ONE ) ;
```

-- Example 10.2.39

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.39");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(008) and
          TYPES.ADA_SQL.LAST_NAME'("Hagan      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Carl      ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A204") and
          TYPES.ADA_SQL.HOME_STATE'("PA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          FOUR ) ;
```

-- Example 10.2.40

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.40");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(009) and
          TYPES.ADA_SQL.LAST_NAME'("Bearman   ") and
          TYPES.ADA_SQL.FIRST_NAME'("Rose     ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("A301") and
          TYPES.ADA_SQL.HOME_STATE'("VA") and
```

# UNCLASSIFIED

```
TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
ONE ) ;
```

-- Example 10.2.41

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.41");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(010) and
        TYPES.ADA_SQL.LAST_NAME'("Thompson    ") and
        TYPES.ADA_SQL.FIRST_NAME'("Paul        ") and
        TYPES.ADA_SQL.GENERAL_ARRAY '("A302") and
        TYPES.ADA_SQL.HOME_STATE'("NC") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
        THREE ) ;
```

-- Example 10.2.42

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.42");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(011) and
        TYPES.ADA_SQL.LAST_NAME'("Bennett    ") and
        TYPES.ADA_SQL.FIRST_NAME'("Nellie    ") and
        TYPES.ADA_SQL.GENERAL_ARRAY '("A303") and
        TYPES.ADA_SQL.HOME_STATE'("PA") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
        THREE ) ;
```

-- Example 10.2.43

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.43");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(012) and
        TYPES.ADA_SQL.LAST_NAME'("Schmidt    ") and
        TYPES.ADA_SQL.FIRST_NAME'("John        ") and
        TYPES.ADA_SQL.GENERAL_ARRAY '("A304") and
        TYPES.ADA_SQL.HOME_STATE'("SC") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
        TWO ) ;
```

-- Example 10.2.44

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.44");

INSERT INTO ( STUDENT ,
```

# UNCLASSIFIED

```
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(013) and
          TYPES.ADA_SQL.LAST_NAME'("Gevarter    ") and
          TYPES.ADA_SQL.FIRST_NAME'("Susan      ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("B101") and
          TYPES.ADA_SQL.HOME_STATE'("NY") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          FOUR ) ;
```

-- Example 10.2.45

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.45");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(014) and
          TYPES.ADA_SQL.LAST_NAME'("Sherman    ") and
          TYPES.ADA_SQL.FIRST_NAME'("Donald    ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("B102") and
          TYPES.ADA_SQL.HOME_STATE'("VA") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          THREE ) ;
```

-- Example 10.2.46

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.46");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(015) and
          TYPES.ADA_SQL.LAST_NAME'("Gorham     ") and
          TYPES.ADA_SQL.FIRST_NAME'("Milton    ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("B103") and
          TYPES.ADA_SQL.HOME_STATE'("WV") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TWO ) ;
```

-- Example 10.2.47

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.47");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and
          TYPES.ADA_SQL.LAST_NAME'("Williams   ") and
          TYPES.ADA_SQL.FIRST_NAME'("Alvin     ") and
          TYPES.ADA_SQL.GENERAL_ARRAY'("B104") and
          TYPES.ADA_SQL.HOME_STATE'("DC") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          ONE ) ;
```

-- Example 10.2.48

# UNCLASSIFIED

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.48");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(017) and
          TYPES.ADA_SQL.LAST_NAME'("Woodliff      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Dorothy    ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("B201") and
          TYPES.ADA_SQL.HOME_STATE'("MD") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          FOUR ) ;
```

-- Example 10.2.49

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.49");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(018) and
          TYPES.ADA_SQL.LAST_NAME'("Ratliff      ") and
          TYPES.ADA_SQL.FIRST_NAME'("Ann        ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("B202") and
          TYPES.ADA_SQL.HOME_STATE'("NY") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          ONE ) ;
```

-- Example 10.2.50

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.50");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(019) and
          TYPES.ADA_SQL.LAST_NAME'("Phung        ") and
          TYPES.ADA_SQL.FIRST_NAME'("Kim        ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("B203") and
          TYPES.ADA_SQL.HOME_STATE'("SC") and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TWO ) ;
```

-- Example 10.2.51

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.51");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(020) and
          TYPES.ADA_SQL.LAST_NAME'("McMurray     ") and
          TYPES.ADA_SQL.FIRST_NAME'("Eric       ") and
          TYPES.ADA_SQL.GENERAL_ARRAY '("B204") and
          TYPES.ADA_SQL.HOME_STATE'("VA") and
```

UNCLASSIFIED

```
TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
ONE ) ;
```

-- Example 10.2.52

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.52");  
  
INSERT INTO ( STUDENT ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(021) and  
          TYPES.ADA_SQL.LAST_NAME'("O'Leary      ") and  
          TYPES.ADA_SQL.FIRST_NAME'("Peggy      ") and  
          TYPES.ADA_SQL.GENERAL_ARRAY '("C101") and  
          TYPES.ADA_SQL.HOME_STATE'("PA") and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and  
          FOUR ) ;
```

-- Example 10.2.53

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.53");  
  
INSERT INTO ( STUDENT ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(022) and  
          TYPES.ADA_SQL.LAST_NAME'("Martin      ") and  
          TYPES.ADA_SQL.FIRST_NAME'("Charlotte ") and  
          TYPES.ADA_SQL.GENERAL_ARRAY '("C102") and  
          TYPES.ADA_SQL.HOME_STATE'("DC") and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and  
          TWO ) ;
```

-- Example 10.2.54

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.54");  
  
INSERT INTO ( STUDENT ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(023) and  
          TYPES.ADA_SQL.LAST_NAME'("O'Day      ") and  
          TYPES.ADA_SQL.FIRST_NAME'("Hilda      ") and  
          TYPES.ADA_SQL.GENERAL_ARRAY '("C103") and  
          TYPES.ADA_SQL.HOME_STATE'("NC") and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          ONE ) ;
```

-- Example 10.2.55

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.55");  
  
INSERT INTO ( STUDENT ,
```

# UNCLASSIFIED

```
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(024) and
        TYPES.ADA_SQL.LAST_NAME'("Martin      ") and
        TYPES.ADA_SQL.FIRST_NAME'("Edward    ") and
        TYPES.ADA_SQL.GENERAL_ARRAY '("C104") and
        TYPES.ADA_SQL.HOME_STATE'("MD") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
        THREE ) ;
```

-- Example 10.2.56

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.56");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(025) and
        TYPES.ADA_SQL.LAST_NAME'("Chateauneuf ") and
        TYPES.ADA_SQL.FIRST_NAME'("Chelsea   ") and
        TYPES.ADA_SQL.GENERAL_ARRAY '("C105") and
        TYPES.ADA_SQL.HOME_STATE'("VA") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
        THREE ) ;
```

-- Example 10.2.57

```
--      select *
--      from STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.2.57");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*' ,
                FROM => STUDENT ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
     "ST_MAJOR ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
```



UNCLASSIFIED

```

GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- ST_ID
  PUT (V_ST_ID, 3);
SET_COL (8); -- ST_NAME
  PUT (V_ST_NAME, V_ST_NAME_INDEX);
SET_COL (22); -- ST_FIRST
  PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
SET_COL (34); -- ST_ROOM
  PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
SET_COL (43); -- ST_STATE
  PUT (V_ST_STATE, V_ST_STATE_INDEX);
SET_COL (53); -- ST_MAJOR
  PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
  PUT (V_ST_YEAR);
NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.2.58

NEW_LINE;
PUT_LINE ("Output of Example 10.2.58");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(001) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.ID_COURSE'(302) and
          TYPES.ADA_SQL.GRADE_POINT'(089.49) and
          TYPES.ADA_SQL.GRADE_POINT'(051.91) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;

-- Example 10.2.59

NEW_LINE;
PUT_LINE ("Output of Example 10.2.59");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(001) and

```

UNCLASSIFIED

```
TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
TYPES.ADA_SQL.ID_COURSE'(303) and
TYPES.ADA_SQL.GRADE_POINT'(077.61) and
TYPES.ADA_SQL.GRADE_POINT'(088.84) and
TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.60

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.60");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(002) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          TYPES.ADA_SQL.ID_COURSE'(103) and
          TYPES.ADA_SQL.GRADE_POINT'(054.38) and
          TYPES.ADA_SQL.GRADE_POINT'(084.77) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.61

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.61");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(003) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.ID_COURSE'(403) and
          TYPES.ADA_SQL.GRADE_POINT'(092.92) and
          TYPES.ADA_SQL.GRADE_POINT'(097.48) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.62

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.62");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(004) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and
          TYPES.ADA_SQL.ID_COURSE'(204) and
          TYPES.ADA_SQL.GRADE_POINT'(071.17) and
          TYPES.ADA_SQL.GRADE_POINT'(070.55) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) );
```

-- Example 10.2.63

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.63");

INSERT INTO ( CLASS ,
```

UNCLASSIFIED

```
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(005) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.ID_COURSE'(503) and
          TYPES.ADA_SQL.GRADE_POINT'(088.83) and
          TYPES.ADA_SQL.GRADE_POINT'(081.12) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.64

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.64");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(006) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and
          TYPES.ADA_SQL.ID_COURSE'(301) and
          TYPES.ADA_SQL.GRADE_POINT'(066.26) and
          TYPES.ADA_SQL.GRADE_POINT'(094.60) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.65

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.65");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(006) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.ID_COURSE'(402) and
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.66

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.66");

INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(007) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.ID_COURSE'(401) and
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.67

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.67");
```

UNCLASSIFIED

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(007) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          TYPES.ADA_SQL.ID_COURSE'(402) and  
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and  
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.68

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.68");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(007) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          TYPES.ADA_SQL.ID_COURSE'(403) and  
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and  
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.69

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.69");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(007) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(503) and  
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and  
          TYPES.ADA_SQL.GRADE_POINT'(100.00) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.70

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.70");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(008) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(502) and  
          TYPES.ADA_SQL.GRADE_POINT'(069.68) and  
          TYPES.ADA_SQL.GRADE_POINT'(056.92) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.71

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.71");
```

UNCLASSIFIED

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(009) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
          TYPES.ADA_SQL.ID_COURSE'(204) and  
          TYPES.ADA_SQL.GRADE_POINT'(055.53) and  
          TYPES.ADA_SQL.GRADE_POINT'(089.81) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.72

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.72");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(010) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and  
          TYPES.ADA_SQL.ID_COURSE'(102) and  
          TYPES.ADA_SQL.GRADE_POINT'(093.72) and  
          TYPES.ADA_SQL.GRADE_POINT'(099.55) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.73

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.73");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(011) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          TYPES.ADA_SQL.ID_COURSE'(401) and  
          TYPES.ADA_SQL.GRADE_POINT'(081.99) and  
          TYPES.ADA_SQL.GRADE_POINT'(076.29) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.74

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.74");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(012) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(501) and  
          TYPES.ADA_SQL.GRADE_POINT'(075.81) and  
          TYPES.ADA_SQL.GRADE_POINT'(083.03) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.75

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.75");
```

UNCLASSIFIED

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(013) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(502) and  
          TYPES.ADA_SQL.GRADE_POINT'(067.36) and  
          TYPES.ADA_SQL.GRADE_POINT'(080.15) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.76

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.76");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(014) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and  
          TYPES.ADA_SQL.ID_COURSE'(302) and  
          TYPES.ADA_SQL.GRADE_POINT'(092.27) and  
          TYPES.ADA_SQL.GRADE_POINT'(082.47) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.77

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.77");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(015) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
          TYPES.ADA_SQL.ID_COURSE'(202) and  
          TYPES.ADA_SQL.GRADE_POINT'(080.75) and  
          TYPES.ADA_SQL.GRADE_POINT'(095.74) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.78

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.78");  
  
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and  
          TYPES.ADA_SQL.ID_COURSE'(101) and  
          TYPES.ADA_SQL.GRADE_POINT'(085.64) and  
          TYPES.ADA_SQL.GRADE_POINT'(078.26) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.79

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.79");
```

UNCLASSIFIED

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and  
          TYPES.ADA_SQL.ID_COURSE'(101) and  
          TYPES.ADA_SQL.GRADE_POINT'(094.59) and  
          TYPES.ADA_SQL.GRADE_POINT'(091.52) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.80

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.80");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
          TYPES.ADA_SQL.ID_COURSE'(204) and  
          TYPES.ADA_SQL.GRADE_POINT'(083.40) and  
          TYPES.ADA_SQL.GRADE_POINT'(094.88) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.81

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.81");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and  
          TYPES.ADA_SQL.ID_COURSE'(302) and  
          TYPES.ADA_SQL.GRADE_POINT'(082.14) and  
          TYPES.ADA_SQL.GRADE_POINT'(087.11) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.82

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.82");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          TYPES.ADA_SQL.ID_COURSE'(403) and  
          TYPES.ADA_SQL.GRADE_POINT'(089.92) and  
          TYPES.ADA_SQL.GRADE_POINT'(097.40) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.83

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.83");
```

UNCLASSIFIED

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(016) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(501) and  
          TYPES.ADA_SQL.GRADE_POINT'(076.86) and  
          TYPES.ADA_SQL.GRADE_POINT'(095.72) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.84

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.84");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(017) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and  
          TYPES.ADA_SQL.ID_COURSE'(401) and  
          TYPES.ADA_SQL.GRADE_POINT'(094.71) and  
          TYPES.ADA_SQL.GRADE_POINT'(063.36) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.85

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.85");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(018) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and  
          TYPES.ADA_SQL.ID_COURSE'(503) and  
          TYPES.ADA_SQL.GRADE_POINT'(092.69) and  
          TYPES.ADA_SQL.GRADE_POINT'(071.69) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.86

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.86");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(019) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
          TYPES.ADA_SQL.ID_COURSE'(201) and  
          TYPES.ADA_SQL.GRADE_POINT'(081.31) and  
          TYPES.ADA_SQL.GRADE_POINT'(095.95) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ,
```

-- Example 10.2.87

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.87");
```



UNCLASSIFIED

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(020) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
          TYPES.ADA_SQL.ID_COURSE'(204) and  
          TYPES.ADA_SQL.GRADE_POINT'(088.28) and  
          TYPES.ADA_SQL.GRADE_POINT'(079.01) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.88

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.88");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(021) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(3) and  
          TYPES.ADA_SQL.ID_COURSE'(303) and  
          TYPES.ADA_SQL.GRADE_POINT'(071.16) and  
          TYPES.ADA_SQL.GRADE_POINT'(074.14) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.89

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.89");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(022) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and  
          TYPES.ADA_SQL.ID_COURSE'(102) and  
          TYPES.ADA_SQL.GRADE_POINT'(058.97) and  
          TYPES.ADA_SQL.GRADE_POINT'(086.58) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.90

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.90");
```

```
INSERT INTO ( CLASS ,  
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(022) and  
          TYPES.ADA_SQL.ID_DEPARTMENT'(2) and  
          TYPES.ADA_SQL.ID_COURSE'(201) and  
          TYPES.ADA_SQL.GRADE_POINT'(081.75) and  
          TYPES.ADA_SQL.GRADE_POINT'(092.97) and  
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.91

```
NEW_LINE;  
PUT_LINE ("Output of Example 10.2.91");
```

# UNCLASSIFIED

```
INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(022) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.ID_COURSE'(503) and
          TYPES.ADA_SQL.GRADE_POINT'(074.49) and
          TYPES.ADA_SQL.GRADE_POINT'(098.30) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.92

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.92");
```

```
INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(023) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(4) and
          TYPES.ADA_SQL.ID_COURSE'(402) and
          TYPES.ADA_SQL.GRADE_POINT'(096.33) and
          TYPES.ADA_SQL.GRADE_POINT'(081.53) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.93

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.93");
```

```
INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(024) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
          TYPES.ADA_SQL.ID_COURSE'(503) and
          TYPES.ADA_SQL.GRADE_POINT'(097.14) and
          TYPES.ADA_SQL.GRADE_POINT'(085.72) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.94

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.94");
```

```
INSERT INTO ( CLASS ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(025) and
          TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
          TYPES.ADA_SQL.ID_COURSE'(101) and
          TYPES.ADA_SQL.GRADE_POINT'(083.58) and
          TYPES.ADA_SQL.GRADE_POINT'(089.16) and
          TYPES.ADA_SQL.GRADE_POINT'(000.00) ) ;
```

-- Example 10.2.95

```
--      select *
--      from CLASS ;
```

UNCLASSIFIED

```

NEW_LINE;
PUT_LINE ("Output of Example 10.2.95");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => CLASS) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("CLASS_STUDENT CLASS_DEPT CLASS_COURSE CLASS_SEM_1 " &
        "CLASS_SEM_2 CLASS_GRADE");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (V_CLASS_DEPT);
    INTO (V_CLASS_COURSE);
    INTO (V_CLASS_SEM_1);
    INTO (V_CLASS_SEM_2);
    INTO (V_CLASS_GRADE);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (15); -- CLASS_DEPT
    PUT (V_CLASS_DEPT, 1);
    SET_COL (26); -- CLASS_COURSE
    PUT (V_CLASS_COURSE, 3);
    SET_COL (39); -- CLASS_SEM_1
    PUT (V_CLASS_SEM_1, 3, 2, 0);
    SET_COL (51); -- CLASS_SEM_2
    PUT (V_CLASS_SEM_2, 3, 2, 0);
    SET_COL (63); -- CLASS_GRADE
    PUT (V_CLASS_GRADE, 3, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

UNCLASSIFIED

-- Example 10.2.96

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.96");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(1) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(1) and
          TYPES.ADA_SQL.YEARLY_INCOME'(20000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(29999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.010) ) ;
```

-- Example 10.2.97

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.97");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(2) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(2) and
          TYPES.ADA_SQL.YEARLY_INCOME'(30000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(34999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.075) ) ;
```

-- Example 10.2.98

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.98");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(3) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(3) and
          TYPES.ADA_SQL.YEARLY_INCOME'(35000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(39999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.050) ) ;
```

-- Example 10.2.99

```
NEW_LINE;
PUT_LINE ("Output of Example 10.2.99");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(4) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(4) and
          TYPES.ADA_SQL.YEARLY_INCOME'(40000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(44999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.035) ) ;
```

-- Example 10.2.100

```
NEW_LINE;
```

# UNCLASSIFIED

```

PUT_LINE ("Output of Example 10.2.100");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(5) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(5) and
          TYPES.ADA_SQL.YEARLY_INCOME'(45000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(49999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.025) ) ;

-- Example 10.2.101

NEW_LINE;
PUT_LINE ("Output of Example 10.2.101");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(6) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(10) and
          TYPES.ADA_SQL.YEARLY_INCOME'(50000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(51999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.020) ) ;

-- Example 10.2.102

NEW_LINE;
PUT_LINE ("Output of Example 10.2.102");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(11) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(15) and
          TYPES.ADA_SQL.YEARLY_INCOME'(52000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(53999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.020) ) ;

-- Example 10.2.103

NEW_LINE;
PUT_LINE ("Output of Example 10.2.103");

INSERT INTO ( SALARY ,
VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(16) and
          TYPES.ADA_SQL.YEARS_EMPLOYED'(20) and
          TYPES.ADA_SQL.YEARLY_INCOME'(54000.00) and
          TYPES.ADA_SQL.YEARLY_INCOME'(55999.00) and
          TYPES.ADA_SQL.SALARY_RAISE'(0.020) ) ;

-- Example 10.2.104

NEW_LINE;
PUT_LINE ("Output of Example 10.2.104");

INSERT INTO ( SALARY ,

```

# UNCLASSIFIED

```

VALUES <= TYPES.ADA_SQL.YEARS_EMPLOYED'(21) and
        TYPES.ADA_SQL.YEARS_EMPLOYED'(99) and
        TYPES.ADA_SQL.YEARLY_INCOME'(56000.00) and
        TYPES.ADA_SQL.YEARLY_INCOME'(60000.00) and
        TYPES.ADA_SQL.SALARY_RAISE'(0.020) );

-- Example 10.2.105

--      select *
--      from SALARY ;

NEW_LINE;
PUT_LINE ("Output of Example 10.2.105");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => SALARY ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("SAL_YEAR  SAL_END  SAL_MIN  SAL_MAX  SAL_RAISE");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_SAL_YEAR);
    INTO (V_SAL_END);
    INTO (V_SAL_MIN);
    INTO (V_SAL_MAX);
    INTO (V_SAL_RAISE);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- SAL_YEAR
    PUT (V_SAL_YEAR, 2);
    SET_COL (11); -- SAL_END
    PUT (V_SAL_END, 2);
    SET_COL (20); -- SAL_MIN
    PUT (V_SAL_MIN, 5, 2, 0);
    SET_COL (30); -- SAL_MAX
    PUT (V_SAL_MAX, 5, 2, 0);
    SET_COL (40); -- SAL_RAISE
    PUT (V_SAL_RAISE, 1, 3, 0);
    NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else

```

# UNCLASSIFIED

```

        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

```

CLOSE ( CURSOR );

```

```

-- Example 10.3.1

```

```

--      select ST_FIRST, ST_NAME, ST_ROOM, ST_YEAR
--      from STUDENT ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.3.1");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_FIRST & ST_NAME & ST_ROOM & ST_YEAR,
        FROM => STUDENT ) );

```

```

OPEN ( CURSOR );

```

```

begin

```

```

    NEW_LINE;
    PUT ("ST_FIRST      ST_NAME      ST_ROOM  ST_YEAR");
    GOT_ONE := 0;

```

```

loop

```

```

    FETCH ( CURSOR );
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

```

```

    SET_COL (1); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (13); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (27); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (36); -- ST_YEAR
    PUT (V_ST_YEAR);

```

```

    NEW_LINE;
end loop;

```

```

exception

```

```

    when NOT_FOUND_ERROR => if GOT_ONE = 0 then

```

# UNCLASSIFIED

```

                PUT_LINE ("EXCEPTION: Not Found Error");
            else
                null;
            end if;
        when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
        when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
    end;

```

```

    CLOSE ( CURSOR );

```

```

-- Example 10.3.2

```

```

--      select PROF_NAME, PROF_SALARY, PROF_YEARS
--      from PROFESSOR ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.3.2");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_NAME & PROF_SALARY & PROF_YEARS,
                FROM => PROFESSOR ) );

```

```

OPEN ( CURSOR );

```

```

begin

```

```

    NEW_LINE;
    PUT_LINE ("PROF_NAME      PROF_SALARY  PROF_YEARS");
    GOT_ONE := 0;

```

```

loop

```

```

    FETCH ( CURSOR );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_SALARY );
    INTO ( V_PROF_YEARS );
    GOT_ONE := GOT_ONE + 1;

```

```

    SET_COL (1); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (15); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    SET_COL (28); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);

```

```

    NEW_LINE;
end loop;

```

```

exception

```

```

    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

```



# UNCLASSIFIED

```

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.4.1

--      select ST_STATE
--      from STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.4.1");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_STATE,
                FROM => STUDENT ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT ("ST_STATE");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_ST_STATE, V_ST_STATE_INDEX);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- ST_STATE
        PUT (V_ST_STATE, V_ST_STATE_INDEX);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                            PUT_LINE ("EXCEPTION: Not Found Error");
                        else
                            null;
                        end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.4.2

--      select distinct ST_STATE
--      from STUDENT ;

```

UNCLASSIFIED

```

NEW_LINE;
PUT_LINE ("Output of Example 10.4.2");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELECT_DISTINCT ( ST_STATE,
        FROM => STUDENT ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_STATE");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.4.3

--      select distinct ST_STATE, ST_YEAR
--      from STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.4.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELECT_DISTINCT ( ST_STATE & ST_YEAR,
        FROM => STUDENT ) );

OPEN ( CURSOR );

begin

```

UNCLASSIFIED

```

NEW_LINE;
PUT ("ST_STATE  ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (11); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.6.1

--      select *
--      from STUDENT
--      where ST_STATE = 'VA' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.1");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => STUDENT,
    WHERE => EQ ( ST_STATE, "VA" ) ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT ("ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
        "ST_MAJOR  ST_YEAR");
    GOT_ONE := 0;

```

# UNCLASSIFIED

```

loop
  FETCH ( CURSOR );
  INTO (V_ST_ID);
  INTO (V_ST_NAME, V_ST_NAME_INDEX);
  INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
  INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
  INTO (V_ST_STATE, V_ST_STATE_INDEX);
  INTO (V_ST_MAJOR);
  INTO (V_ST_YEAR);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- ST_ID
  PUT (V_ST_ID, 3);
  SET_COL (8); -- ST_NAME
  PUT (V_ST_NAME, V_ST_NAME_INDEX);
  SET_COL (22); -- ST_FIRST
  PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
  SET_COL (34); -- ST_ROOM
  PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
  SET_COL (43); -- ST_STATE
  PUT (V_ST_STATE, V_ST_STATE_INDEX);
  SET_COL (53); -- ST_MAJOR
  PUT (V_ST_MAJOR, 1);
  SET_COL (63); -- ST_YEAR
  PUT (V_ST_YEAR);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.6.2

--      select *
--      from STUDENT
--      where ST_NAME = 'McGinn      ' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.2");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*' ,

```

UNCLASSIFIED

```

FROM => STUDENT,
  WHERE => EQ ( ST_NAME, "McGinn      " ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;

  exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
      PUT_LINE ("EXCEPTION: Not Found Error");
    else
      null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
  end;

  CLOSE ( CURSOR );

```

# UNCLASSIFIED

```
-- Example 10.6.3

--      select *
--      from PROFESSOR
--      where PROF_YEARS = 1 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM => PROFESSOR,
        WHERE => EQ ( PROF_YEARS, 1 ) ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
```

# UNCLASSIFIED

```

        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

```

CLOSE ( CURSOR );
```

```
-- Example 10.6.4
```

```
--      select *
--      from PROFESSOR
--      where PROF_YEARS <> 1 ;
```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.6.4");
```

```

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => PROFESSOR,
    WHERE => NE ( PROF_YEARS, 1 ) ) );
```

```

OPEN ( CURSOR );
```

```
begin
```

```

    NEW_LINE;
    PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
    GOT_ONE := 0;
```

```
loop
```

```

    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;
```

```

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
```

# UNCLASSIFIED

```

        PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.6.5

--      select *
--      from PROFESSOR
--      where PROF_YEARS > 1 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.5");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
        FROM => PROFESSOR,
        WHERE => PROF_YEARS > 1 ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO ( V_PROF_ID );
        INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
        INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
        INTO ( V_PROF_DEPT );
        INTO ( V_PROF_YEARS );
        INTO ( V_PROF_SALARY );
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- PROF_ID
        PUT (V_PROF_ID, 2);
        SET_COL (10); -- PROF_NAME

```



UNCLASSIFIED

```

        PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
        PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
        PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
        PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
        PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                                PUT_LINE ("EXCEPTION: Not Found Error");
                                else
                                    null;
                                end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.6.6

--      select *
--      from PROFESSOR
--      where PROF_YEARS >= 4 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.6");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM => PROFESSOR,
        WHERE => PROF_YEARS >= 4 ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );

```

UNCLASSIFIED

```

    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.6.7

--      select *
--      from PROFESSOR
--      where PROF_YEARS < 4 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.7");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*' ,
        FROM => PROFESSOR,
        WHERE => PROF_YEARS < 4 ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &

```

UNCLASSIFIED

```

        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.6.8

--      select *
--      from PROFESSOR
--      where PROF_YEARS <= 3 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.8");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',

```

UNCLASSIFIED

```

FROM => PROFESSOR,
WHERE => PROF_YEARS <= 3 ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
          "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
FETCH ( CURSOR );
INTO ( V_PROF_ID );
INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
INTO ( V_PROF_DEPT );
INTO ( V_PROF_YEARS );
INTO ( V_PROF_SALARY );
GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                        PUT_LINE ("EXCEPTION: Not Found Error");
                        else
                          null;
                        end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.6.9

--      select *
```

UNCLASSIFIED

```
--      from CLASS
--      where CLASS_SEM_2 < CLASS_SEM_1 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.6.9");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => CLASS,
              WHERE => CLASS_SEM_2 < CLASS_SEM_1 ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_STUDENT CLASS_DEPT CLASS_COURSE CLASS_SEM_1 " &
           "CLASS_SEM_2 CLASS_GRADE");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (V_CLASS_DEPT);
    INTO (V_CLASS_COURSE);
    INTO (V_CLASS_SEM_1);
    INTO (V_CLASS_SEM_2);
    INTO (V_CLASS_GRADE);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (15); -- CLASS_DEPT
    PUT (V_CLASS_DEPT, 1);
    SET_COL (26); -- CLASS_COURSE
    PUT (V_CLASS_COURSE, 3);
    SET_COL (39); -- CLASS_SEM_1
    PUT (V_CLASS_SEM_1, 3, 2, 0);
    SET_COL (51); -- CLASS_SEM_2
    PUT (V_CLASS_SEM_2, 3, 2, 0);
    SET_COL (63); -- CLASS_GRADE
    PUT (V_CLASS_GRADE, 3, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
```

UNCLASSIFIED

```
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

```
CLOSE ( CURSOR );
```

```
-- Example 10.7.1
```

```
--      select *
--      from STUDENT
--      where ST_STATE = 'VA' and ST_YEAR = 1 ;
```

```
NEW_LINE;
PUT_LINE ("Output of Example 10.7.1");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM   => STUDENT,
        WHERE  => EQ ( ST_STATE, "VA" )
        AND    EQ ( ST_YEAR, ONE ) ) );
```

```
OPEN ( CURSOR );
```

```
begin
```

```
NEW_LINE;
PUT ("ST_ID ST_NAME      ST_FIRST ST_ROOM ST_STATE " &
    "ST_MAJOR ST_YEAR");
GOT_ONE := 0;
```

```
loop
```

```
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;
```

```
    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
```

UNCLASSIFIED

```

        SET_COL (63);  -- ST_YEAR
        PUT (V_ST_YEAR);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.7.2

--      select *
--      from STUDENT
--      where ST_STATE = 'NC' or ST_STATE = 'SC' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.7.2");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
        FROM => STUDENT,
        WHERE => EQ ( ST_STATE, "NC" )
            OR      EQ ( ST_STATE, "SC" ) ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
        "ST_MAJOR  ST_YEAR");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_ST_ID);
        INTO (V_ST_NAME, V_ST_NAME_INDEX);
        INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
        INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
        INTO (V_ST_STATE, V_ST_STATE_INDEX);
        INTO (V_ST_MAJOR);
        INTO (V_ST_YEAR);
        GOT_ONE := GOT_ONE + 1;
    end loop;
end;

```

# UNCLASSIFIED

```

SET_COL (1); -- ST_ID
PUT (V_ST_ID, 3);
SET_COL (8); -- ST_NAME
PUT (V_ST_NAME, V_ST_NAME_INDEX);
SET_COL (22); -- ST_FIRST
PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
SET_COL (34); -- ST_ROOM
PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
SET_COL (43); -- ST_STATE
PUT (V_ST_STATE, V_ST_STATE_INDEX);
SET_COL (53); -- ST_MAJOR
PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
PUT (V_ST_YEAR);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.7.3

--      select *
--      from STUDENT
--      where ( ST_STATE = 'NC' or ST_STATE = 'SC' )
--      and ST_YEAR = 2 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.7.3");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
        FROM => STUDENT,
        WHERE => ( EQ ( ST_STATE, "NC" )
            OR
            EQ ( ST_STATE, "SC" ) )
        AND
            EQ ( ST_YEAR, TWO ) ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_ID ST_NAME ST_FIRST ST_ROOM ST_STATE " &

```



## UNCLASSIFIED

```

        "ST_MAJOR ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.7.4

--      select *
--      from PROFESSOR
--      where PROF_YEARS <= 4
--            and PROF_SALARY > 33000.00 ;

NEW_LINE;

```

UNCLASSIFIED

```

PUT_LINE ("Output of Example 10.7.4");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => PROFESSOR,
    WHERE => PROF_YEARS <= 4
      AND      PROF_SALARY > 33000.00 ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
    "PROF_YEARS  PROF_SALARY");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

UNCLASSIFIED

CLOSE ( CURSOR );

-- Example 10.7.5

```
--      select *
--      from STUDENT
--      where ( ST_STATE = 'VA' and ST_YEAR = 1 )
--            or ( ( ST_STATE = 'NC' or ST_STATE = 'SC' )
--                  and ST_YEAR = 2 )
--            or ( ST_STATE = 'MD' and ST_YEAR = 3 )
--            or ST_YEAR = 4
--            or ST_STATE = 'DC' ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.7.5");

DECLAR ( CURSOR , CURSOR\_FOR =>

```
  SELEC ( '*',
  FROM => STUDENT,
  WHERE => ( EQ ( ST_STATE, "VA" ) and EQ ( ST_YEAR, ONE ) )
    OR      ( ( EQ ( ST_STATE, "NC" ) or EQ ( ST_STATE, "SC" ) )
              and EQ ( ST_YEAR, TWO ) )
    OR      ( ( EQ ( ST_STATE, "MD" ) and EQ ( ST_YEAR, THREE ) ) )
    OR      ( ( EQ ( ST_YEAR, FOUR ) ) )
    OR      ( ( EQ ( ST_STATE, "DC" ) ) ) ) ) ;
```

OPEN ( CURSOR );

begin

NEW\_LINE;

```
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
```

GOT\_ONE := 0;

loop

```
  FETCH ( CURSOR );
  INTO (V_ST_ID);
  INTO (V_ST_NAME, V_ST_NAME_INDEX);
  INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
  INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
  INTO (V_ST_STATE, V_ST_STATE_INDEX);
  INTO (V_ST_MAJOR);
  INTO (V_ST_YEAR);
  GOT_ONE := GOT_ONE + 1;
```

SET\_COL (1); -- ST\_ID

PUT (V\_ST\_ID, 3);

SET\_COL (8); -- ST\_NAME

PUT (V\_ST\_NAME, V\_ST\_NAME\_INDEX);

SET\_COL (22); -- ST\_FIRST

PUT (V\_ST\_FIRST, V\_ST\_FIRST\_INDEX);

# UNCLASSIFIED

```

SET_COL (34); -- ST_ROOM
PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
SET_COL (43); -- ST_STATE
PUT (V_ST_STATE, V_ST_STATE_INDEX);
SET_COL (53); -- ST_MAJOR
PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
PUT (V_ST_YEAR);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.8.1

--      select *
--      from PROFESSOR
--      where PROF_SALARY >= 35000.00
--      and PROF_SALARY <= 45000.00 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.8.1");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => PROFESSOR,
    WHERE => PROF_SALARY >= 35000.00
    AND    PROF_SALARY <= 45000.00 ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
    "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );

```

UNCLASSIFIED

```

    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.8.2

--      select *
--      from PROFESSOR
--      where PROF_SALARY
--            between 35000.00 and 45000.00 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.8.2");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
        FROM => PROFESSOR,
        WHERE => BETWEEN ( PROF_SALARY, 35000.00 and 45000.00 ) ) );

OPEN ( CURSOR );

begin

```

# UNCLASSIFIED

```

NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
          "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTC ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.9.1

--      select *
--      from STUDENT
--      where ST_STATE = 'VA'
--            or  ST_STATE = 'MD'
--            or  ST_STATE = 'DC' ;

NEW_LINE;

```

UNCLASSIFIED

```

PUT_LINE ("Output of Example 10.9.1");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*' ,
    FROM => STUDENT,
    WHERE => EQ ( ST_STATE, "VA" )
      OR      EQ ( ST_STATE, "MD" )
      OR      EQ ( ST_STATE, "DC" ) ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;

```

# UNCLASSIFIED

```

when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

```

CLOSE ( CURSOR );

```

```

-- Example 10.9.2

```

```

--      select *
--      from STUDENT
--      where ST_STATE in ( 'VA', 'MD', 'DC' ) ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.9.2");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM => STUDENT,
        WHERE => IS_IN ( ST_STATE, "VA" or "MD" or "DC" ) ) );

```

```

OPEN ( CURSOR );

```

```

begin

```

```

NEW_LINE;
PUT ("ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
     "ST_MAJOR  ST_YEAR");
GOT_ONE := 0;

```

```

loop

```

```

    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

```

```

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);

```



UNCLASSIFIED

```

        SET_COL (63); -- ST_YEAR
        PUT (V_ST_YEAR);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.11.1

--      select ST_NAME
--      from STUDENT
--      where ST_NAME like 'S%';

NEW_LINE;
PUT_LINE ("Output of Example 10.11.1");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( ST_NAME,
        FROM => STUDENT,
        WHERE => LIKE ( ST_NAME, "S*****" ) ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT ("ST_NAME");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_ST_NAME, V_ST_NAME_INDEX);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- ST_NAME
        PUT (V_ST_NAME, V_ST_NAME_INDEX);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");

```

# UNCLASSIFIED

```

                                else
                                    null;
                                end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.11.2

--      select ST_NAME, ST_ROOM
--      from STUDENT
--      where ST_ROOM like 'A%' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.11.2");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_NAME & ST_ROOM,
        FROM   => STUDENT,
        WHERE  => LIKE ( ST_ROOM, "A%" ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_NAME           ST_ROOM");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");

```

UNCLASSIFIED

```

end;

CLOSE ( CURSOR );

-- Example 10.11.3

--      select ST_NAME, ST_ROOM
--      from STUDENT
--      where ST_ROOM like 'A___' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.11.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_NAME & ST_ROOM,
        FROM => STUDENT,
        WHERE => LIKE ( ST_ROOM, "A___" ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_NAME           ST_ROOM");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.11.4

```

UNCLASSIFIED

```
--      select ST_NAME, ST_ROOM
--      from STUDENT
--      where ST_ROOM like '_101' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.11.4");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_NAME & ST_ROOM,
        FROM => STUDENT,
        WHERE => LIKE ( ST_ROOM, "_101" ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_NAME           ST_ROOM");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.12.1

--      select *
--      from STUDENT
--      where not (ST_STATE = 'VA' ) ;

NEW_LINE;
```

UNCLASSIFIED

```

PUT_LINE ("Output of Example 10.12.1");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => STUDENT,
    WHERE => NOT ( EQ ( ST_STATE, "VA" ) ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");

```

UNCLASSIFIED

```

end;

CLOSE ( CURSOR );

-- Example 10.12.2

--      select *
--      from PROFESSOR
--      where not ( PROF_YEARS <= 4
--      and PROF_SALARY > 33000.00 ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.12.2");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM => PROFESSOR,
        WHERE => NOT ( PROF_YEARS <= 4
        AND      PROF_SALARY > 33000.00 ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;

```

UNCLASSIFIED

```

end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.12.3

--      select *
--      from PROFESSOR
--      where PROF_SALARY
--            not between 35000.00 and 45000.00 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.12.3");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => PROFESSOR,
    WHERE => NOT ( BETWEEN ( PROF_SALARY, 35000.00 and 45000.00 ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
    "PROF_YEARS  PROF_SALARY");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);

```

UNCLASSIFIED

```

SET_COL (24); -- PROF_FIRST
  PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
  PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
  PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.12.4

--      select *
--      from STUDENT
--      where ST_STATE not in ( 'VA', 'MD', 'DC' ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.12.4");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
  FROM => STUDENT,
  WHERE => NOT ( IS_IN ( ST_STATE, "VA" or "MD" or "DC" ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);

```



# UNCLASSIFIED

```

INTO (V_ST_STATE, V_ST_STATE_INDEX);
INTO (V_ST_MAJOR);
INTO (V_ST_YEAR);
GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- ST_ID
  PUT (V_ST_ID, 3);
SET_COL (8); -- ST_NAME
  PUT (V_ST_NAME, V_ST_NAME_INDEX);
SET_COL (22); -- ST_FIRST
  PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
SET_COL (34); -- ST_ROOM
  PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
SET_COL (43); -- ST_STATE
  PUT (V_ST_STATE, V_ST_STATE_INDEX);
SET_COL (53); -- ST_MAJOR
  PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
  PUT (V_ST_YEAR);
NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.12.5

--      select ST_NAME, ST_ROOM
--      from STUDENT
--      where ST_ROOM not like 'A%' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.12.5");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC  ( ST_NAME & ST_ROOM,
    FROM  => STUDENT,
    WHERE => NOT ( LIKE ( ST_ROOM, "A%" ) ) ) );

OPEN ( CURSOR );

begin

```

# UNCLASSIFIED

```

NEW_LINE;
PUT ("ST_NAME                ST_ROOM");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.13.1

--      select PROF_NAME, PROF_SALARY * 1.10
--      from PROFESSOR ;

NEW_LINE;
PUT_LINE ("Output of Example 10.13.1");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( PROF_NAME & ( PROF_SALARY * 1.10 ),
    FROM => PROFESSOR ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_NAME      PROF_SALARY * 1.10");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );

```

UNCLASSIFIED

```

    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (15); -- PROF_SALARY * 1.10
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.13.2

--      select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS ;
--

NEW_LINE;
PUT_LINE ("Output of Example 10.13.2");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("CLASS_STUDENT      CLASS_SEM_1 + CLASS_SEM_2 / 2");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_CLASS_STUDENT);
        INTO (V_CLASS_SEM_1);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- CLASS_STUDENT
        PUT (V_CLASS_STUDENT, 3);
        SET_COL (19); -- CLASS_SEM_1 + CLASS_SEM_2 / 2

```

UNCLASSIFIED

```

        PUT (V_CLASS_SEM_1, 3, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.13.3

--      select PROF_NAME, PROF_SALARY
--      from PROFESSOR
--      where PROF_SALARY > PROF_YEARS * 10000.00 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.13.3");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( PROF_NAME & PROF_SALARY,
    FROM   => PROFESSOR,
    WHERE  => PROF_SALARY >
        ( CONVERT_TO.TYPES.YEARLY_INCOME ( PROF_YEARS ) * 10000.00 ) ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_NAME      PROF_SALARY");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
        INTO ( V_PROF_SALARY );
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- PROF_NAME
        PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
        SET_COL (15); -- PROF_SALARY
        PUT (V_PROF_SALARY, 5, 2, 0);
        NEW_LINE;
    end loop;

```

# UNCLASSIFIED

```

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;

  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.13.4

--      select PROF_NAME, PROF_SALARY * 1.10
--      from PROFESSOR
--      where PROF_SALARY * 1.10 < PROF_YEARS * 10000.00 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.13.4");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( PROF_NAME & ( PROF_SALARY * 1.10 ),
    FROM => PROFESSOR,
    WHERE => PROF_SALARY * 1.10 <
      CONVERT_TO.TYPES.YEARLY_INCOME ( PROF_YEARS ) * 10000.00 ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("PROF_NAME      PROF_SALARY * 1.10");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (15); -- PROF_SALARY * 1.10
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else

```

# UNCLASSIFIED

```

                null;
            end if;
        when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
        when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
    end;

```

```

    CLOSE ( CURSOR );

```

```

-- Example 10.14.1

```

```

--      select count (*)
--      from STUDENT ;
--

```

```

begin
    NEW_LINE;
    PUT_LINE ("Output of Example 10.14.1");

```

```

    SELEC ( COUNT ('*'),
            FROM => STUDENT );
            INTO (COUNT_RESULT);

```

```

    NEW_LINE;
    PUT ("COUNT");
    SET_COL (1); -- COUNT
    PUT (COUNT_RESULT, 3);
    NEW_LINE;

```

```

exception
    when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

```

-- Example 10.14.2

```

```

--      select count (*)
--      from STUDENT
--      where ST_ROOM like 'A%' ;

```

```

begin
    NEW_LINE;
    PUT_LINE ("Output of Example 10.14.2");

```

```

    SELEC ( COUNT ('*'),
            FROM => STUDENT,
            WHERE => LIKE ( ST_ROOM, "A%" ) );
            INTO (COUNT_RESULT);

```

```

    NEW_LINE;
    PUT ("COUNT");

```

UNCLASSIFIED

```

SET_COL (1); -- COUNT
PUT (COUNT_RESULT, 3);
NEW_LINE;

exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

-- Example 10.14.3

```

--      select count (*)
--      from STUDENT
--      where ST_STATE in ( 'DC', 'VA', 'MD' ) ;

begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.14.3");

  SELEC ( COUNT ('*'),
    FROM => STUDENT,
    WHERE => IS_IN (ST_STATE, "DC" or "VA" or "MD" ) );
  INTO (COUNT_RESULT);

  NEW_LINE;
  PUT ("COUNT");
  SET_COL (1); -- COUNT
  PUT (COUNT_RESULT, 3);
  NEW_LINE;
```

```

exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

-- Example 10.14.5

```

--      select count (*)
--      from STUDENT
--      where ST_STATE = 'VA' ;

begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.14.5");

  SELEC ( COUNT ('*'),
    FROM => STUDENT,
    WHERE => EQ ( ST_STATE, "VA" ) );
  INTO (COUNT_RESULT);
```

# UNCLASSIFIED

```

NEW_LINE;
PUT ("COUNT");
SET_COL (1); -- COUNT
    PUT (COUNT_RESULT, 3);
NEW_LINE;

exception
    when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

-- Example 10.14.7

--      select min (PROF_SALARY), max (PROF_SALARY)
--      from PROFESSOR ;

begin
    NEW_LINE;
    PUT_LINE ("Output of Example 10.14.7");

    SELEC ( min (PROF_SALARY) & max (PROF_SALARY),
        FROM => PROFESSOR );
        INTO ( MIN_SALARY );
        INTO ( MAX_SALARY );

    NEW_LINE;
    PUT_LINE ("MINIMUM SALARY      MAXIMUM SALARY");
    SET_COL (1); -- MIN SALARY
        PUT (MIN_SALARY, 7, 2, 0);
    SET_COL (20); -- MAX SALARY
        PUT (MAX_SALARY, 7, 2, 0);
    NEW_LINE;

exception
    when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

-- Example 10.14.8

--      select sum (PROF_SALARY)
--      from PROFESSOR ;

begin
    NEW_LINE;
    PUT_LINE ("Output of Example 10.14.8");

    SELEC ( CONVERT_TO.TYPES.TOTAL_INCOME (sum (PROF_SALARY)),

```



# UNCLASSIFIED

```

FROM => PROFESSOR );
    INTO ( SUM_SALARY );

NEW_LINE;
PUT_LINE ("SALARY");
SET_COL (1); -- SUM SALARY
    PUT (SUM_SALARY, 7, 2, 0);
NEW_LINE;

exception
    when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

-- Example 10.14.9

--      select avg (CLASS_SEM_1), avg (CLASS_SEM_2)
--      from CLASS
--      where CLASS_STUDENT = 016 ;

begin
    NEW_LINE;
    PUT_LINE ("Output of Example 10.14.9");

    SELEC ( avg (CLASS_SEM_1) & avg (CLASS_SEM_2),
        FROM => CLASS,
        WHERE => EQ (CLASS_STUDENT, 016) );
        INTO (AVG_SEM_1);
        INTO (AVG_SEM_2);

    NEW_LINE;
    PUT_LINE ("AVERAGE CLASS_SEM_1      AVERAGE CLASS_SEM_2");
    SET_COL (1); -- AVG_SEM_1
        PUT (AVG_SEM_1, 3, 2, 0);
    SET_COL (25); -- AVG_SEM_2
        PUT (AVG_SEM_2, 3, 2, 0);
    NEW_LINE;

exception
    when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

-- Example 10.14.10

--      select count (*), sum (PROF_SALARY), avg (PROF_SALARY),
--      min (PROF_SALARY), max (PROF_SALARY)
--      from PROFESSOR ;

```

UNCLASSIFIED

```

begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.14.10");

  SELEC ( COUNT ('*') & CONVERT_TO.TYPES.TOTAL_INCOME (sum (PROF_SALARY)) &
    avg (PROF_SALARY) & min (PROF_SALARY) & max (PROF_SALARY),
    FROM => PROFESSOR );
    INTO ( COUNT_RESULT );
    INTO ( SUM_SALARY );
    INTO ( AVG_SALARY );
    INTO ( MIN_SALARY );
    INTO ( MAX_SALARY );

  NEW_LINE;
  PUT_LINE ("COUNT SALARY SUM          AVERAGE    MINIMUM    MAXIMUM");
  SET_COL (1); -- COUNT
    PUT (COUNT_RESULT, 3);
  SET_COL (15); -- SUM SALARY
    PUT (SUM_SALARY, 9, 2, 0);
  SET_COL (28); -- AVG SALARY
    PUT (AVG_SALARY, 7, 2, 0);
  SET_COL (39); -- MIN SALARY
    PUT (MIN_SALARY, 7, 2, 0);
  SET_COL (50); -- MAX SALARY
    PUT (MAX_SALARY, 7, 2, 0);
  NEW_LINE;

exception
  when NOT_FOUND_ERROR => PUT_LINE ("EXCEPTION: Not Found Error");
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

-- Example 10.15.1

--      select *
--      from STUDENT
--      order by ST_NAME ;

NEW_LINE;
PUT_LINE ("Output of Example 10.15.1");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => STUDENT ),
    ORDER_BY => ST_NAME );

OPEN ( CURSOR );

begin
  NEW_LINE;

```

UNCLASSIFIED

```

PUT ("ST_ID ST_NAME ST_FIRST ST_ROOM ST_STATE " &
     "ST_MAJOR ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.15.2

--      select PROF_NAME, PROF_SALARY
--      from PROFESSOR
--      order by PROF_SALARY desc ;

NEW_LINE;

```

UNCLASSIFIED

```

PUT_LINE ("Output of Example 10.15.2");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_NAME & PROF_SALARY,
                FROM => PROFESSOR ),
        ORDER_BY => DESC (PROF_SALARY) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_NAME      PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (15); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.15.3

--      select ST_NAME, ST_YEAR, ST_MAJOR
--      from STUDENT
--      order by ST_YEAR desc, ST_MAJOR ;

NEW_LINE;
PUT_LINE ("Output of Example 10.15.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( ST_NAME & ST_YEAR & ST_MAJOR,
                FROM => STUDENT ),

```

UNCLASSIFIED

```

ORDER_BY => DESC (ST_YEAR) & ST_MAJOR );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_NAME      ST_YEAR  ST_MAJOR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_YEAR);
    INTO (V_ST_MAJOR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (15); -- ST_YEAR
    PUT (V_ST_YEAR);
    SET_COL (24); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.15.4

--      select ST_NAME, ST_YEAR, ST_MAJOR
--      from STUDENT
--      order by ST_YEAR desc, ST_MAJOR asc ;

NEW_LINE;
PUT_LINE ("Output of Example 10.15.4");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( ST_NAME & ST_YEAR & ST_MAJOR,
        FROM => STUDENT ),
        ORDER_BY => DESC (ST_YEAR) & ASC (ST_MAJOR) );

```

# UNCLASSIFIED

```

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_NAME          ST_YEAR  ST_MAJOR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_YEAR);
    INTO (V_ST_MAJOR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (15); -- ST_YEAR
    PUT (V_ST_YEAR);
    SET_COL (24); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.15.5

--      select CLASS_DEPT, CLASS_COURSE, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2,
--      CLASS_STUDENT
--      from CLASS
--      where ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 >= 90.00
--      and CLASS_SEM_2 - CLASS_SEM_1 >= 5.00 )
--      or CLASS_SEM_1 = 100.00
--      or CLASS_SEM_2 = 100.00
--      order by CLASS_SEM_2 desc, CLASS_SEM_1 desc, CLASS_DEPT asc,
--      CLASS_COURSE asc ;

NEW_LINE;
PUT_LINE ("Output of Example 10.15.5");

DECLAR ( CURSOR , CURSOR_FOR =>

```

UNCLASSIFIED

```

SELECT ( CLASS_DEPT & CLASS_COURSE &
        ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) & CLASS_STUDENT &
        CLASS_SEM_1 & CLASS_SEM_2,
FROM => CLASS,
WHERE => ( ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 >= 90.00
          AND CLASS_SEM_2 - CLASS_SEM_1 >= 5.00 )
        OR
          EQ (CLASS_SEM_1, 100.00)
        OR
          EQ (CLASS_SEM_2, 100.00) ),
ORDER_BY => DESC (CLASS_SEM_2) & DESC (CLASS_SEM_1) & ASC (CLASS_DEPT) &
          ASC (CLASS_COURSE) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("CLASS_DEPT CLASS_COURSE CLASS_SEM_1 + CLASS_SEM_2 / 2 " &
          "CLASS_STUDENT");
GOT_ONE := 0;

loop
FETCH ( CURSOR );
INTO (V_CLASS_DEPT);
INTO (V_CLASS_COURSE);
INTO (AVG_SEM_1);
INTO (V_CLASS_STUDENT);
GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- CLASS_DEPT
PUT (V_CLASS_DEPT, 1);
SET_COL (13); -- CLASS_COURSE
PUT (V_CLASS_COURSE, 3);
SET_COL (28); -- AVG_SEM_1
PUT (AVG_SEM_1, 3, 2, 0);
SET_COL (60); -- CLASS_STUDENT
PUT (V_CLASS_STUDENT, 3);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                        PUT_LINE ("EXCEPTION: Not Found Error");
                        else
                          null;
                        end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.16.1

```

# UNCLASSIFIED

```
--      select CLASS_STUDENT, avg (CLASS_SEM_1), avg (CLASS_SEM_2)
--      from CLASS
--      group by CLASS_STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.16.1");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & avg (CLASS_SEM_1) & avg (CLASS_SEM_2),
    FROM => CLASS,
    GROUP_BY => CLASS_STUDENT ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_STUDENT      AVG_SEM_1      AVG_SEM_2");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (AVG_SEM_1);
    INTO (AVG_SEM_2);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (18); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    SET_COL (30); -- AVG_SEM_2
    PUT (AVG_SEM_2, 3, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.16.2

--      select CLASS_STUDENT, avg (CLASS_SEM_1), avg (CLASS_SEM_2)
--      from CLASS
```



UNCLASSIFIED

```
--          where CLASS_DEPT = 3
--          group by CLASS_STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.16.2");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & avg (CLASS_SEM_1) & avg (CLASS_SEM_2),
    FROM => CLASS,
    WHERE => EQ (CLASS_DEPT, 3),
    GROUP_BY => CLASS_STUDENT ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("CLASS_STUDENT      AVG_SEM_1      AVG_SEM_2");
GOT_ONE := 0;

loop
  FETCH ( CURSOR );
  INTO (V_CLASS_STUDENT);
  INTO (AVG_SEM_1);
  INTO (AVG_SEM_2);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- CLASS_STUDENT
  PUT (V_CLASS_STUDENT, 3);
  SET_COL (18); -- AVG_SEM_1
  PUT (AVG_SEM_1, 3, 2, 0);
  SET_COL (31); -- AVG_SEM_2
  PUT (AVG_SEM_2, 3, 2, 0);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.16.3

--          select CLASS_STUDENT, CLASS_DEPT, avg (CLASS_SEM_1), avg (CLASS_SEM_2)
--          from CLASS
```

# UNCLASSIFIED

```
--          group by CLASS_DEPT, CLASS_STUDENT ;

NEW_LINE;
PUT_LINE ("Output of Example 10.16.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( CLASS_STUDENT & CLASS_DEPT & avg (CLASS_SEM_1) & avg (CLASS_SEM_2),
        FROM => CLASS,
        GROUP_BY => CLASS_DEPT & CLASS_STUDENT ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("CLASS_STUDENT   CLASS_DEPT   AVG_SEM_1   AVG_SEM_2");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (V_CLASS_DEPT);
    INTO (AVG_SEM_1);
    INTO (AVG_SEM_2);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (17); -- CLASS_DEPT
    PUT (V_CLASS_DEPT, 1);
    SET_COL (30); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    SET_COL (43); -- AVG_SEM_2
    PUT (AVG_SEM_2, 3, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.16.4

--          select ST_STATE, ST_MAJOR, ST_YEAR, count(*)
```

UNCLASSIFIED

```
--      from STUDENT
--      group by ST_STATE, ST_MAJOR, ST_YEAR ;

NEW_LINE;
PUT_LINE ("Output of Example 10.16.4");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( ST_STATE & ST_MAJOR & ST_YEAR & count ('*'),
    FROM => STUDENT,
    GROUP_BY => ST_STATE & ST_MAJOR & ST_YEAR ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_STATE   ST_MAJOR   ST_YEAR   COUNT");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    INTO (COUNT_RESULT);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (12); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (23); -- ST_YEAR
    PUT (V_ST_YEAR);
    SET_COL (33); -- COUNT
    PUT (COUNT_RESULT, 3);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;

  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.17.1
```

# UNCLASSIFIED

```
--      select max (PROF_SALARY)
--      from PROFESSOR ;

begin
  NEW_LINE;
  PUT_LINE ("Output of Example 10.17.1");

  SELEC ( max (PROF_SALARY),
    FROM => PROFESSOR ) ;
    INTO ( V_PROF_SALARY );

  NEW_LINE;
  PUT_LINE ("MAX_PROF_SALARY");
  SET_COL (1); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
  NEW_LINE;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

-- Example 10.17.2

--      select PROF_FIRST, PROF_NAME, PROF_SALARY
--      from PROFESSOR
--      where PROF_SALARY = 50000.00 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.2");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( PROF_FIRST & PROF_NAME & PROF_SALARY,
    FROM => PROFESSOR,
    WHERE => EQ ( PROF_SALARY, 50000.00 ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("PROF_FIRST  PROF_NAME      PROF_SALARY");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
```

UNCLASSIFIED

```

    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (13); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (27); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.17.3

--      select PROF_FIRST, PROF_NAME, PROF_SALARY
--      from PROFESSOR
--      where PROF_SALARY =
--            ( select max (PROF_SALARY)
--              from PROFESSOR ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.3");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( PROF_FIRST & PROF_NAME & PROF_SALARY,
    FROM => PROFESSOR,
    WHERE => EQ ( PROF_SALARY,
    SELEC ( max (PROF_SALARY),
    FROM => PROFESSOR ) ) ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_FIRST  PROF_NAME      PROF_SALARY");
    GOT_ONE := 0;

    loop

```

UNCLASSIFIED

```

FETCH ( CURSOR );
INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
INTO ( V_PROF_SALARY );
GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (13); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (27); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.17.4

--      select PROF_ID, PROF_SALARY
--      from PROFESSOR
--      where PROF_SALARY >
--            ( select min ( PROF_SALARY )
--              from PROFESSOR )
--      and PROF_SALARY <
--            ( select max ( PROF_SALARY )
--              from PROFESSOR ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.4");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( PROF_ID & PROF_SALARY,
    FROM => PROFESSOR,
    WHERE => PROF_SALARY >
      SELEC ( min ( PROF_SALARY ),
        FROM => PROFESSOR )
    AND      PROF_SALARY <
      SELEC ( max ( PROF_SALARY ),
        FROM => PROFESSOR ) ) ) ;

```

# UNCLASSIFIED

```

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("PROF_ID  PROF_SALARY");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.17.5

--      select PROF_ID, PROF_SALARY, PROF_YEARS
--      from PROFESSOR
--      where PROF_SALARY >
--            ( select min ( PROF_SALARY )
--              from PROFESSOR
--              where PROF_YEARS >
--                ( select avg ( PROF_YEARS )
--                  from PROFESSOR ) ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.5");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( PROF_ID & PROF_SALARY & PROF_YEARS,
    FROM => PROFESSOR,
    WHERE => PROF_SALARY >
      SELEC ( min ( PROF_SALARY ),

```

# UNCLASSIFIED

```

FROM => PROFESSOR,
WHERE => PROF_YEARS >
SELEC ( avg ( PROF_YEARS ),
FROM => PROFESSOR ) ) );

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID PROF_SALARY PROF_YEARS");
GOT_ONE := 0;

loop
FETCH ( CURSOR );
INTO ( V_PROF_ID );
INTO ( V_PROF_SALARY );
INTO ( V_PROF_YEARS );
GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
SET_COL (23); -- PROF_YEARS
PUT (V_PROF_YEARS, 2);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
PUT_LINE ("EXCEPTION: Not Found Error");
else
null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.17.6

-- select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
-- from CLASS
-- where CLASS_STUDENT = any
-- ( select CLASS_STUDENT
-- from CLASS
-- group by CLASS_STUDENT
-- having count (*) > 2 ) ;

NEW_LINE;

```



# UNCLASSIFIED

```

PUT_LINE ("Output of Example 10.17.6");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
  FROM => CLASS,
  WHERE => IS_IN ( CLASS_STUDENT,
  SELEC ( CLASS_STUDENT,
  FROM => CLASS,
  GROUP_BY => CLASS_STUDENT,
  HAVING => count ( '*' ) > 2 ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_STUDENT   CLASS_SEM_1 + CLASS_SEM_2 / 2");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (AVG_SEM_1);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (17); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.17.10

--      select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS
--      where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 >=
--      ( select avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--      from CLASS

```

# UNCLASSIFIED

```

--          where CLASS_STUDENT = any
--          ( select CLASS_STUDENT
--            from CLASS
--              group by CLASS_STUDENT
--                having count (*) > 2 ) )
-- and CLASS_STUDENT = any
--   ( select CLASS_STUDENT
--     from CLASS
--       group by CLASS_STUDENT
--         having count (*) > 2 ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.10");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS,
    WHERE => ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 >=
      SELEC ( avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
        FROM => CLASS,
        WHERE => IS_IN ( CLASS_STUDENT,
          SELEC ( CLASS_STUDENT,
            FROM => CLASS,
            GROUP_BY => CLASS_STUDENT,
            HAVING => count ( '*' ) > 2 ) ) )
        AND IS_IN ( CLASS_STUDENT,
          SELEC ( CLASS_STUDENT,
            FROM => CLASS,
            GROUP_BY => CLASS_STUDENT,
            HAVING => count ( '*' ) > 2 ) ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_STUDENT  CLASS_SEM_1 + CLASS_SEM_2 / 2");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (AVG_SEM_1);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (17); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    NEW_LINE;
  end loop;

```

# UNCLASSIFIED

```
exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;
```

```
CLOSE ( CURSOR );
```

```
-- Example 10.17.11
```

```
--      select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS
--      where CLASS_STUDENT in
--      ( select CLASS_STUDENT
--        from CLASS
--        group by CLASS_STUDENT
--        having count (*) > 2 ) ;
```

```
NEW_LINE;
```

```
PUT_LINE ("Output of Example 10.17.11");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
  FROM => CLASS,
  WHERE => IS_IN ( CLASS_STUDENT,
  SELEC ( CLASS_STUDENT,
  FROM => CLASS,
  GROUP_BY => CLASS_STUDENT,
  HAVING => count ( '*' ) > 2 ) ) ) ) ;
```

```
OPEN ( CURSOR );
```

```
begin
```

```
NEW_LINE;
```

```
PUT_LINE ("CLASS_STUDENT  CLASS_SEM_1 + CLASS_SEM_2 / 2");
```

```
GOT_ONE := 0;
```

```
loop
```

```
  FETCH ( CURSOR );
```

```
  INTO (V_CLASS_STUDENT);
```

```
  INTO (AVG_SEM_1);
```

```
  GOT_ONE := GOT_ONE + 1;
```

```
  SET_COL (1); -- CLASS_STUDENT
```

```
  PUT (V_CLASS_STUDENT, 3);
```

```
  SET_COL (17); -- AVG_SEM_1
```

```
  PUT (AVG_SEM_1, 3, 2, 0);
```

# UNCLASSIFIED

```

NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                           PUT_LINE ("EXCEPTION: Not Found Error");
                           else
                             null;
                           end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

-- Example 10.17.12

```

--      select CLASS_STUDENT, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS
--      where CLASS_STUDENT not in
--      ( select CLASS_STUDENT
--        from CLASS
--        group by CLASS_STUDENT
--        having count (*) <= 2 ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.17.12");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00),
  FROM => CLASS,
  WHERE => NOT IS_IN ( CLASS_STUDENT,
  SELEC ( CLASS_STUDENT,
  FROM => CLASS,
  GROUP_BY => CLASS_STUDENT,
  HAVING => count ('*') <= 2 ) ) ) ;

OPEN ( CURSOR );

```

```

begin
  NEW_LINE;
  PUT_LINE ("CLASS_STUDENT  CLASS_SEM_1 + CLASS_SEM_2 / 2");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (AVG_SEM_1);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT

```

# UNCLASSIFIED

```

        PUT (V_CLASS_STUDENT, 3);
        SET_COL (17); -- AVG_SEM_1
        PUT (AVG_SEM_1, 3, 2, 0);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR     => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.18.1

--      select COURSE_DEPT, sum ( COURSE_PROF)
--      from COURSE
--      group by COURSE_DEPT
--      having sum ( COURSE_PROF ) > 10 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.18.1");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( COURSE_DEPT & sum ( COURSE_PROF ),
        FROM => COURSE,
        GROUP_BY => COURSE_DEPT,
        HAVING => sum ( COURSE_PROF ) > 10 ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("COURSE_DEPT    COURSE_PROF ");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_COURSE_DEPT);
        INTO (V_COURSE_PROF);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- COURSE_DEPT
        PUT (V_COURSE_DEPT, 1);
        SET_COL (15); -- COURSE_PROF
        PUT (V_COURSE_PROF, 3);
    end loop;
end;

```

# UNCLASSIFIED

```

    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.18.2

--      select COURSE_DEPT, count (*)
--      from COURSE
--      group by COURSE_DEPT
--      having count (*) > 3 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.18.2");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( COURSE_DEPT & count ('*'),
    FROM => COURSE,
    GROUP_BY => COURSE_DEPT,
    HAVING => count ('*') > 3 ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("COURSE_DEPT    COUNT");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_COURSE_DEPT);
        INTO (COUNT_RESULT);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- COURSE_DEPT
        PUT (V_COURSE_DEPT, 1);
        SET_COL (15); -- COUNT_RESULT
        PUT (COUNT_RESULT, 3);
        NEW_LINE;
    end loop;

```

# UNCLASSIFIED

```

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR      => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.18.3

--      select ST_STATE, ST_MAJOR, count (*)
--      from STUDENT
--      where ST_STATE in ( 'VA', 'DC', 'MD', 'NC', 'PA' )
--      group by ST_STATE, ST_MAJOR
--      having avg ( ST_MAJOR ) > 2 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.18.3");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( ST_STATE & ST_MAJOR & count ('*'),
    FROM => STUDENT,
    WHERE => IS_IN ( ST_STATE, "VA" or "DC" or "MD" or "NC" or "PA" ),
    GROUP_BY => ST_STATE & ST_MAJOR,
    HAVING => avg ( ST_MAJOR ) > 2 ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_STATE ST_MAJOR COUNT");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (COUNT_RESULT);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (11); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (21); -- COUNT_RESULT
    PUT (COUNT_RESULT, 3);
    NEW_LINE;
  end loop;

```

# UNCLASSIFIED

```

end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

-- Example 10.18.4

--      select CLASS_COURSE, avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--      from CLASS
--      where CLASS_DEPT = 2 or CLASS_DEPT = 4
--      group by CLASS_COURSE
--      having avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 ) >
--      ( select avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--        from CLASS ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.18.4");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_COURSE & avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS,
    WHERE => EQ ( CLASS_DEPT, 2 )
    OR      EQ ( CLASS_DEPT, 4 ),
    GROUP_BY => CLASS_COURSE,
    HAVING => avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) >
    SELEC ( avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
      FROM => CLASS ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_COURSE  AVG CLASS_SEM_1 + CLASS_SEM_2 / 2"),
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_COURSE);
    INTO (AVG_SEM_1);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1);  -- CLASS_COURSE

```



# UNCLASSIFIED

```

        PUT (V_CLASS_COURSE, 3);
        SET_COL (15); -- AVG_SEM_1
        PUT (AVG_SEM_1, 3, 2, 0);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR     => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.18.6

--      select CLASS_COURSE, avg ( CLASS_SEM_1 ), avg ( CLASS_SEM_2 )
--      from CLASS
--      group by CLASS_COURSE
--      having CLASS_COURSE = any
--      ( select CLASS_COURSE
--        from CLASS
--        where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--              ( select max ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--                from CLASS )
--        or ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--              ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--                from CLASS ) ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.18.6");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( CLASS_COURSE & avg ( CLASS_SEM_1 ) & avg ( CLASS_SEM_2 ),
    FROM => CLASS,
    GROUP_BY => CLASS_COURSE,
    HAVING => IS_IN ( CLASS_COURSE,
    SELEC ( CLASS_COURSE,
    FROM => CLASS,
    WHERE => EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
    SELEC ( max ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS ) )
    OR
    EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
    SELEC ( min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
    FROM => CLASS ) ) ) ) ) ;

OPEN ( CURSOR );

```

UNCLASSIFIED

```

begin
  NEW_LINE;
  PUT_LINE ("CLASS_COURSE   AVG SEM_1   AVG SEM_2");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_COURSE);
    INTO (AVG_SEM_1);
    INTO (AVG_SEM_2);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_COURSE
    PUT (V_CLASS_COURSE, 3);
    SET_COL (16); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    SET_COL (28); -- AVG_SEM_2
    PUT (AVG_SEM_2, 3, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.18.7

--      select CLASS_STUDENT, CLASS_SEM_2
--      from CLASS
--      where CLASS_COURSE =
--            ( select CLASS_COURSE
--              from CLASS
--              group by CLASS_COURSE
--              having avg ( CLASS_SEM_2 ) =
--                    ( select max ( avg ( CLASS_SEM_2 ) )
--                      from CLASS
--                      group by CLASS_COURSE ) ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.18.7");

DECLAR ( CURSOR , CURSOR_FOR =>

```

UNCLASSIFIED

```

        SELEC ( CLASS_STUDENT & CLASS_SEM_2,
        FROM => CLASS,
        WHERE => EQ ( CLASS_COURSE,
        SELEC ( CLASS_COURSE,
        FROM => CLASS,
        GROUP_BY => CLASS_COURSE,
        HAVING => EQ ( avg ( CLASS_SEM_2 ),
        SELEC ( max ( avg ( CLASS_SEM_2 ) ),
        FROM => CLASS,
        GROUP_BY => CLASS_COURSE ) ) ) ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("CLASS_STUDENT    CLASS_SEM_2");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_CLASS_STUDENT);
        INTO (V_CLASS_SEM_2);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- CLASS_STUDENT
        PUT (V_CLASS_STUDENT, 3);
        SET_COL (17); -- CLASS_SEM_2
        PUT (V_CLASS_SEM_2, 3, 2, 0);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.19.1

--      select *
--      from DEPARTMENT, PROFESSOR ;

NEW_LINE;
PUT_LINE ("Output of Example 10.19.1");

```

# UNCLASSIFIED

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*' ,
    FROM => DEPARTMENT & PROFESSOR ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("DEPARTMENT      PROFESSOR");
  PUT_LINE ("ID      DESC      ID NAME      FIRST      DEPT  YEARS  SALARY");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO ( V_DEPT_ID );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- DEPT_ID
    PUT (V_DEPT_ID, 1);
    SET_COL (6); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    SET_COL (16); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (20); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (34); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (46); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (52); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
  end loop;

  exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
      PUT_LINE ("EXCEPTION: Not Found Error");
    else
      null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");

```

## UNCLASSIFIED

end;

CLOSE ( CURSOR );

--Example 10.19.2

```
--      select PROF_FIRST, PROF_NAME, DEPT_DESC
--      from PROFESSOR, DEPARTMENT
--      where PROF_DEPT = DEPT_ID ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.19.2");

```
DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_FIRST & PROF_NAME & DEPT_DESC,
        FROM => PROFESSOR & DEPARTMENT,
        WHERE => EQ ( PROF_DEPT, DEPT_ID ) ) ) ;
```

OPEN ( CURSOR );

begin

NEW\_LINE;

PUT\_LINE ("PROF\_FIRST    PROF\_NAME        DEPT\_DESC");

GOT\_ONE := 0;

loop

FETCH ( CURSOR );

INTO ( V\_PROF\_FIRST, V\_PROF\_FIRST\_INDEX );

INTO ( V\_PROF\_NAME, V\_PROF\_NAME\_INDEX );

INTO ( V\_DEPT\_DESC, V\_DEPT\_DESC\_INDEX );

GOT\_ONE := GOT\_ONE + 1;

SET\_COL (1); -- PROF\_FIRST

PUT (V\_PROF\_FIRST, V\_PROF\_FIRST\_INDEX);

SET\_COL (14); -- PROF\_NAME

PUT (V\_PROF\_NAME, V\_PROF\_NAME\_INDEX);

SET\_COL (28); -- DEPT\_DESC

PUT (STRING(V\_DEPT\_DESC (1..V\_DEPT\_DESC\_INDEX)));

NEW\_LINE;

end loop;

exception

when NOT\_FOUND\_ERROR => if GOT\_ONE = 0 then

PUT\_LINE ("EXCEPTION: Not Found Error");

else

null;

end if;

when NO\_UPDATE\_ERROR => PUT\_LINE ("EXCEPTION: No Update Error");

when UNIQUE\_ERROR => PUT\_LINE ("EXCEPTION: Unique Error");

end;

# UNCLASSIFIED

CLOSE ( CURSOR );

--Example 10.19.3

```
--      select DEPT_DESC, COURSE_DESC, PROF_FIRST, PROF_NAME
--      from PROFESSOR, DEPARTMENT, COURSE
--      where COURSE_DEPT = DEPT_ID
--      and COURSE_PROF = PROF_ID ;
```

NEW\_LINE;

PUT\_LINE ("Output of Example 10.19.3");

```
DECLAR ( CURSOR , CURSOR_FOR =>
      SELEC ( DEPT_DESC & COURSE_DESC & PROF_FIRST & PROF_NAME,
      FROM => PROFESSOR & DEPARTMENT & COURSE,
      WHERE => EQ ( COURSE_DEPT, DEPT_ID )
      AND      EQ ( COURSE_PROF, PROF_ID ) ) ) ;
```

OPEN ( CURSOR );

begin

NEW\_LINE;

PUT\_LINE ("DEPT\_DESC COURSE\_DESC PROF\_FIRST PROF\_NAME");

GOT\_ONE := 0;

loop

FETCH ( CURSOR );

INTO ( V\_DEPT\_DESC, V\_DEPT\_DESC\_INDEX );

INTO ( V\_COURSE\_DESC, V\_COURSE\_DESC\_INDEX );

INTO ( V\_PROF\_FIRST, V\_PROF\_FIRST\_INDEX );

INTO ( V\_PROF\_NAME, V\_PROF\_NAME\_INDEX );

GOT\_ONE := GOT\_ONE + 1;

SET\_COL (1); -- DEPT\_DESC

PUT (STRING(V\_DEPT\_DESC (1..V\_DEPT\_DESC\_INDEX)));

SET\_COL (12); -- COURSE\_DESC

PUT (V\_COURSE\_DESC, V\_COURSE\_DESC\_INDEX);

SET\_COL (34); -- PROF\_FIRST

PUT (V\_PROF\_FIRST, V\_PROF\_FIRST\_INDEX);

SET\_COL (46); -- PROF\_NAME

PUT (V\_PROF\_NAME, V\_PROF\_NAME\_INDEX);

NEW\_LINE;

end loop;

exception

when NOT\_FOUND\_ERROR => if GOT\_ONE = 0 then

PUT\_LINE ("EXCEPTION: Not Found Error");

else

null;

end if;

when NO\_UPDATE\_ERROR => PUT\_LINE ("EXCEPTION: No Update Error");

UNCLASSIFIED

```

when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

```

CLOSE ( CURSOR );

```

```

--Example 10.19.4

```

```

--      select DEPT_DESC, COURSE_DESC, PROF_FIRST, PROF_NAME
--      from PROFESSOR, DEPARTMENT, COURSE
--      where COURSE_DEPT = DEPT_ID
--      and COURSE_PROF = PROF_ID
--      order by DEPT_ID, COURSE_ID ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.19.4");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
      SELEC ( DEPT_DESC & COURSE_DESC & PROF_FIRST & PROF_NAME &
              DEPT_ID & COURSE_ID ,
      FROM => PROFESSOR & DEPARTMENT & COURSE,
      WHERE => EQ ( COURSE_DEPT, DEPT_ID )
      AND      EQ ( COURSE_PROF, PROF_ID ) ),
      ORDER_BY => DEPT_ID & COURSE_ID ) ;

```

```

OPEN ( CURSOR );

```

```

begin

```

```

NEW_LINE;
PUT_LINE ("DEPT_DESC  COURSE_DESC          PROF_FIRST  PROF_NAME");
GOT_ONE := 0;

```

```

loop

```

```

      FETCH ( CURSOR );
      INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
      INTO ( V_COURSE_DESC, V_COURSE_DESC_INDEX );
      INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
      INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
      GOT_ONE := GOT_ONE + 1;

```

```

      SET_COL (1); -- DEPT_DESC
      PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
      SET_COL (12); -- COURSE_DESC
      PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
      SET_COL (34); -- PROF_FIRST
      PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
      SET_COL (46); -- PROF_NAME
      PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
      NEW_LINE;
end loop;

```

```

exception

```

# UNCLASSIFIED

```

when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

```

CLOSE ( CURSOR );

```

--Example 10.19.5

```

--      select DEPT_DESC, COURSE_DESC, PROF_FIRST, PROF_NAME
--      from PROFESSOR, DEPARTMENT, COURSE
--      where COURSE_DEPT = DEPT_ID
--      and COURSE_PROF = PROF_ID
--      order by DEPT_DESC, COURSE_DESC ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.19.5");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
    SELE ( DEPT_DESC & COURSE_DESC & PROF_FIRST & PROF_NAME,
        FROM => PROFESSOR & DEPARTMENT & COURSE,
        WHERE => EQ ( COURSE_DEPT, DEPT_ID )
        AND      EQ ( COURSE_PROF, PROF_ID ) ),
    ORDER_BY => DEPT_DESC & COURSE_DESC ) ;

```

```

OPEN ( CURSOR );

```

begin

```

NEW_LINE;
PUT_LINE ("DEPT_DESC  COURSE_DESC          PROF_FIRST  PROF_NAME");
GOT_ONE := 0;

```

loop

```

    FETCH ( CURSOR );
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    GOT_ONE := GOT_ONE + 1;

```

```

    SET_COL (1); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    SET_COL (12); -- COURSE_DESC
    PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    SET_COL (34); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (46); -- PROF_NAME

```



# UNCLASSIFIED

```

        PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

--Example 10.19.6

```

--      select DEPT_DESC, COURSE_DESC, PROF_NAME, ST_NAME
--      from DEPARTMENT, COURSE, PROFESSOR, STUDENT, CLASS
--      where CLASS_STUDENT = any
--          ( select CLASS_STUDENT
--            from CLASS
--            where CLASS_SEM_1 =
--              ( select max ( CLASS_SEM_1 )
--                from CLASS )
--            or   CLASS_SEM_1 =
--              ( select min ( CLASS_SEM_1 )
--                from CLASS )
--            or   CLASS_SEM_2 =
--              ( select max ( CLASS_SEM_2 )
--                from CLASS )
--            or   CLASS_SEM_2 =
--              ( select min ( CLASS_SEM_2 )
--                from CLASS ) )
--      and CLASS_STUDENT = ST_ID
--      and CLASS_DEPT = DEPT_ID
--      and CLASS_COURSE = COURSE_ID
--      and COURSE_PROF = PROF_ID
--      group by ST_NAME, COURSE_DESC, DEPT_DESC, PROF_NAME
--      order by DEPT_DESC, COURSE_DESC, PROF_NAME, ST_NAME ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.19.6");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( DEPT_DESC & COURSE_DESC & PROF_NAME & ST_NAME,
    FROM => DEPARTMENT & COURSE & PROFESSOR & STUDENT & CLASS,
    WHERE => IS_IN ( CLASS_STUDENT,
        SELEC ( CLASS_STUDENT,

```

# UNCLASSIFIED

```

FROM => CLASS,
  WHERE => EQ ( CLASS_SEM_1,
    SELEC ( max ( CLASS_SEM_1 ),
      FROM => CLASS ) )
  OR EQ ( CLASS_SEM_1,
    SELEC ( min ( CLASS_SEM_1 ),
      FROM => CLASS ) )
  OR EQ ( CLASS_SEM_2,
    SELEC ( max ( CLASS_SEM_2 ),
      FROM => CLASS ) )
  OR EQ ( CLASS_SEM_2,
    SELEC ( min ( CLASS_SEM_2 ),
      FROM => CLASS ) ) ) )
AND EQ ( CLASS_STUDENT, ST_ID )
AND EQ ( CLASS_DEPT, DEPT_ID )
AND EQ ( CLASS_COURSE, COURSE_ID )
AND EQ ( COURSE_PROF, PROF_ID ),
  GROUP_BY => ST_NAME & COURSE_DESC & DEPT_DESC & PROF_NAME ),
  ORDER_BY => DEPT_DESC & COURSE_DESC & PROF_NAME & ST_NAME );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("DEPT_DESC      JURSE_DESC          PROF_NAME      ST_NAME");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_DEPT_DESC, V_DEPT_DESC_INDEX);
    INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    INTO (V_PROF_NAME, V_PROF_NAME_INDEX);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    SET_COL (13); -- COURSE_DESC
    PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    SET_COL (35); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (49); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;

```

# UNCLASSIFIED

```

                                end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.19.7

--      select PROF_NAME, PROF_YEARS, SAL_YEAR, SAL_END, PROF_SALARY,
--             SAL_MIN, SAL_MAX
--      from PROFESSOR, SALARY
--      where PROF_YEARS >= SAL_YEAR
--            and PROF_YEARS <= SAL_END ;

NEW_LINE;
PUT_LINE ("Output of Example 10.19.7");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_NAME & PROF_YEARS & SAL_YEAR & SAL_END & PROF_SALARY &
                SAL_MIN & SAL_MAX,
        FROM => PROFESSOR & SALARY,
        WHERE => PROF_YEARS >= SAL_YEAR
        AND PROF_YEARS <= SAL_END ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_NAME      PROF_YEARS  SAL_YEAR  SAL_END  PROF_SALARY  " &
        "SAL_MIN  SAL_MAX");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_YEARS );
    INTO ( V_SAL_YEAR );
    INTO ( V_SAL_END );
    INTO ( V_PROF_SALARY );
    INTO ( V_SAL_MIN );
    INTO ( V_SAL_MAX );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (15); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (27); -- SAL_YEAR
    PUT (V_SAL_YEAR, 2);
    SET_COL (37); -- SAL_END

```

# UNCLASSIFIED

```

        PUT (V_SAL_END, 2);
SET_COL (46); -- PROF_SALARY
        PUT (V_PROF_SALARY, 5, 2, 0);
SET_COL (59); -- SAL_MIN
        PUT (V_SAL_MIN, 5, 2, 0);
SET_COL (69); -- SAL_MAX
        PUT (V_SAL_MAX, 5, 2, 0);
NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.19.8

--      select PROF_NAME, PROF_SALARY, PROF_YEARS, SAL_YEAR, SAL_END,
--              SAL_MIN, SAL_MAX
--      from PROFESSOR, SALARY
--      where PROF_SALARY between SAL_MIN and SAL_MAX ;

NEW_LINE;
PUT_LINE ("Output of Example 10.19.8");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( PROF_NAME & PROF_SALARY & PROF_YEARS & SAL_YEAR & SAL_END &
                SAL_MIN & SAL_MAX,
        FROM => PROFESSOR & SALARY,
        WHERE => BETWEEN ( PROF_SALARY, SAL_MIN and SAL_MAX ) ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_NAME      PROF_SALARY  PROF_YEARS  SAL_YEAR  SAL_END  " &
              "SAL_MIN      SAL_MAX");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
        INTO ( V_PROF_SALARY );
        INTO ( V_PROF_YEARS );

```

UNCLASSIFIED

```

    INTO (V_SAL_YEAR);
    INTO (V_SAL_END);
    INTO (V_SAL_MIN);
    INTO (V_SAL_MAX);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (15); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    SET_COL (28); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (40); -- SAL_YEAR
    PUT (V_SAL_YEAR, 2);
    SET_COL (50); -- SAL_END
    PUT (V_SAL_END, 2);
    SET_COL (59); -- SAL_MIN
    PUT (V_SAL_MIN, 5, 2, 0);
    SET_COL (69); -- SAL_MAX
    PUT (V_SAL_MAX, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.20.1

--      select DEPARTMENT.DEPT_DESC, COURSE.COURSE_DESC, PROFESSOR.PROF_NAME,
--      STUDENT.ST_NAME
--      from DEPARTMENT, COURSE, PROFESSOR, STUDENT, CLASS
--      where STUDENT.ST_ID = any
--      ( select CLASS_STUDENT
--        from CLASS
--        group by CLASS_STUDENT
--        having COUNT (*) >= 4 )
--      and CLASS.CLASS_STUDENT = STUDENT.ST_ID
--      and CLASS.CLASS_COURSE = COURSE.COURSE_ID
--      and COURSE.COURSE_DEPT = DEPARTMENT.DEPT_ID
--      and COURSE.COURSE_PROF = PROFESSOR.PROF_ID ;

NEW_LINE;

```

# UNCLASSIFIED

```

PUT_LINE ("Output of Example 10.20.1");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( DEPARTMENT.DEPT_DESC & COURSE.COURSE_DESC & PROFESSOR.PROF_NAME &
    STUDENT.ST_NAME,
  FROM => DEPARTMENT & COURSE & PROFESSOR & STUDENT & CLASS,
  WHERE => IS_IN ( STUDENT.ST_ID,
    SELEC ( CLASS_STUDENT,
      FROM => CLASS,
      GROUP_BY => CLASS_STUDENT,
      HAVING => COUNT ( '*' ) >= 4 ) )
  AND EQ ( CLASS.CLASS_STUDENT, STUDENT.ST_ID )
  AND EQ ( CLASS.CLASS_COURSE, COURSE.COURSE_ID )
  AND EQ ( COURSE.COURSE_DEPT, DEPARTMENT.DEPT_ID )
  AND EQ ( COURSE.COURSE_PROF, PROFESSOR.PROF_ID ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("DEPT_DESC  COURSE_DESC          PROF_NAME      ST_NAME");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    GOT_ONE := GOT_ONE + 1;
    INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
    INTO ( V_COURSE_DESC, V_COURSE_DESC_INDEX );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_ST_NAME, V_ST_NAME_INDEX );

    SET_COL (1); -- DEPT_DESC
    PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
    SET_COL (12); -- COURSE_DESC
    PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    SET_COL (34); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (48); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;

  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

## UNCLASSIFIED

```
CLOSE ( CURSOR );
```

```
--Example 10.20.2
```

```
--      select D.DEPT_DESC, C.COURSE_DESC, P.PROF_NAME, S.ST_NAME
--      from DEPARTMENT D, COURSE C, PROFESSOR P, STUDENT S, CLASS CL
--      where S.ST_ID = any
--            ( select CLASS_STUDENT
--              from CLASS
--              group by CLASS_STUDENT
--              having COUNT (*) >= 4 )
--      and CL.CLASS_STUDENT = S.ST_ID
--      and CL.CLASS_COURSE = C.COURSE_ID
--      and C.COURSE_DEPT = D.DEPT_ID
--      and C.COURSE_PROF = P.PROF_ID ;
```

```
NEW_LINE;
```

```
PUT_LINE ("Output of Example 10.20.2");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( D.DEPT_DESC & C.COURSE_DESC & P.PROF_NAME & S.ST_NAME,
    FROM => D.DEPARTMENT & C.COURSE & P.PROFESSOR & S.STUDENT & CL.CLASS,
    WHERE => IS_IN ( S.ST_ID,
      SELEC ( CLASS_STUDENT,
        FROM => CLASS,
        GROUP_BY => CLASS_STUDENT,
        HAVING => COUNT ( '*' ) >= 4 ) )
    AND EQ ( CL.CLASS_STUDENT, S.ST_ID )
    AND EQ ( CL.CLASS_COURSE, C.COURSE_ID )
    AND EQ ( C.COURSE_DEPT, D.DEPT_ID )
    AND EQ ( C.COURSE_PROF, P.PROF_ID ) ) ) ;
```

```
OPEN ( CURSOR );
```

```
begin
```

```
NEW_LINE;
```

```
PUT_LINE ("DEPT_DESC  COURSE_DESC          PROF_NAME      ST_NAME");
```

```
GOT_ONE := 0;
```

```
loop
```

```
  FETCH ( CURSOR );
```

```
  GOT_ONE := GOT_ONE + 1;
```

```
  INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
```

```
  INTO ( V_COURSE_DESC, V_COURSE_DESC_INDEX );
```

```
  INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
```

```
  INTO ( V_ST_NAME, V_ST_NAME_INDEX );
```

```
  SET_COL (1); -- DEPT_DESC
```

```
  PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
```

```
  SET_COL (12); -- COURSE_DESC
```

```
  PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
```

# UNCLASSIFIED

```

SET_COL (34); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (48); -- ST_NAME
PUT (V_ST_NAME, V_ST_NAME_INDEX);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.21.1

--      select X.PROF_NAME, X.PROF_SALARY
--      from PROFESSOR X, PROFESSOR Y
--      where X.PROF_SALARY >= Y.PROF_SALARY
--      and Y.PROF_NAME = 'Hall          ' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.21.1");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( X.PROF_NAME & X.PROF_SALARY,
    FROM => X.PROFESSOR & Y.PROFESSOR,
    WHERE => X.PROF_SALARY >= Y.PROF_SALARY
    AND EQ ( Y.PROF_NAME, "Hall          " ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_NAME      PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (15); -- PROF_SALARY

```



UNCLASSIFIED

```

        PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR     => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.21.2

--      select X.PROF_FIRST, X.PROF_NAME, X.PROF_SALARY, DEPT_DESC, COURSE_DESC
--      from PROFESSOR X, PROFESSOR Y, DEPARTMENT, COURSE
--      where X.PROF_SALARY >= Y.PROF_SALARY
--            and Y.PROF_NAME = 'Hall
--            and X.PROF_ID = COURSE_PROF
--            and COURSE_DEPT = DEPT_ID ;

NEW_LINE;
PUT_LINE ("Output of Example 10.21.2");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( X.PROF_FIRST & X.PROF_NAME & X.PROF_SALARY & DEPT_DESC &
        COURSE_DESC,
    FROM => X.PROFESSOR & Y.PROFESSOR & DEPARTMENT & COURSE,
    WHERE => X.PROF_SALARY >= Y.PROF_SALARY
        AND EQ ( Y.PROF_NAME, "Hall
            " )
        AND EQ ( X.PROF_ID, COURSE_PROF )
        AND EQ ( COURSE_DEPT, DEPT_ID ) ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_FIRST  PROF_NAME      PROF_SALARY  DEPT_DESC  COURSE_DESC");
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
        INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
        INTO ( V_PROF_SALARY );
        INTO ( V_DEPT_DESC, V_DEPT_DESC_INDEX );
        INTO (V_COURSE_DESC, V_COURSE_DESC_INDEX);
    end loop;

```

# UNCLASSIFIED

```

GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- PROF_FIRST
PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (13); -- PROF_NAME
PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (27); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
SET_COL (40); -- DEPT_DESC
PUT (STRING(V_DEPT_DESC (1..V_DEPT_DESC_INDEX)));
SET_COL (51); -- COURSE_DESC
PUT (V_COURSE_DESC, V_COURSE_DESC_INDEX);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.22.5

--      select avg (PROF_SALARY)
--      from PROFESSOR, COURSE
--      where COURSE_HOURS > 3
--      and PROF_ID = COURSE_PROF ;

NEW_LINE;
PUT_LINE ("Output of Example 10.22.5");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( avg (PROF_SALARY),
    FROM => PROFESSOR & COURSE,
    WHERE => COURSE_HOURS > THREE
    AND EQ ( PROF_ID, COURSE_PROF ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("AVG PROF_SALARY");
GOT_ONE := 0;

loop

```

UNCLASSIFIED

```

    FETCH ( CURSOR );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR      => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

--Example 10.22.6

```

--      select PROF_SALARY, PROF_ID, COURSE_HOURS, COURSE_ID
--      from PROFESSOR, COURSE
--      where COURSE_HOURS > 3
--      and PROF_ID = COURSE_PROF ;

NEW_LINE;
PUT_LINE ("Output of Example 10.22.6");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( PROF_SALARY & PROF_ID & COURSE_HOURS & COURSE_ID,
    FROM => PROFESSOR & COURSE,
    WHERE => COURSE_HOURS > THREE
    AND EQ ( PROF_ID, COURSE_PROF ) ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_SALARY  PROF_ID  COURSE_HOURS  COURSE_ID");
    GOT_ONE := 0;

```

```

    loop
        FETCH ( CURSOR );
        INTO ( V_PROF_SALARY );
        INTO ( V_PROF_ID );
        INTO (V_COURSE_HOURS);
        INTO (V_COURSE_ID);
        GOT_ONE := GOT_ONE + 1;

```

# UNCLASSIFIED

```

SET_COL (1); -- PROF_SALARY
PUT (V_PROF_SALARY, 5, 2, 0);
SET_COL (14); -- PROF_ID
PUT (V_PROF_ID, 2);
SET_COL (23); -- COURSE_HOURS
PUT (V_COURSE_HOURS);
SET_COL (37); -- COURSE_ID
PUT (V_COURSE_ID, 3);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.23.1

--      insert into STUDENT
--      values
--      ( 026, 'Brenner      ', 'Samuel      ', 'A101', 'CA', 5, 1 );

NEW_LINE;
PUT_LINE ("Output of Example 10.23.1");

INSERT INTO ( STUDENT ,
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(026) and
        TYPES.ADA_SQL.LAST_NAME'("Brenner      ") and
        TYPES.ADA_SQL.FIRST_NAME'("Samuel      ") and
        TYPES.ADA_SQL.GENERAL_ARRAY'("A101") and
        TYPES.ADA_SQL.HOME_STATE'("CA") and
        TYPES.ADA_SQL.ID_DEPARTMENT'(5) and
        ONE ) ;

--Example 10.23.2

--      insert into STUDENT
--      ( ST_YEAR, ST_STATE, ST_NAME )
--      values
--      ( 1, 'AK', 'Mamout      ' ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.23.2");

```

UNCLASSIFIED

```

INSERT INTO ( STUDENT ( ST_ID & ST_YEAR & ST_STATE & ST_NAME & ST_FIRST &
                ST_MAJOR & ST_ROOM),
VALUES <= TYPES.ADA_SQL.ID_STUDENT'(99) and
                ONE and
                TYPES.ADA_SQL.HOME_STATE'("AK") and
                TYPES.ADA_SQL.LAST_NAME'("Mamout      ") and
                TYPES.ADA_SQL.FIRST_NAME'("              ") and
                TYPES.ADA_SQL.ID_DEPARTMENT'(1) and
                TYPES.ADA_SQL.GENERAL_ARRAY '("              ") ) ;

```

--Example 10.23.3

```

--      select *
--      from STUDENT
--      where ST_ID not between 1 and 25 ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.23.3");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*' ,
        FROM => STUDENT,
        WHERE => NOT BETWEEN ( ST_ID, 1 and 25 ) ) );

```

```

OPEN ( CURSOR );

```

begin

```

NEW_LINE;
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
     "ST_MAJOR  ST_YEAR");
GOT_ONE := 0;

```

loop

```

    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);

```

UNCLASSIFIED

```

SET_COL (43); -- ST_STATE
PUT (V_ST_STATE, V_ST_STATE_INDEX);
SET_COL (53); -- ST_MAJOR
PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
PUT (V_ST_YEAR);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.23.4

--      select *
--      from STUDENT
--      where ST_NAME like ('M%') ;

NEW_LINE;
PUT_LINE ("Output of Example 10.23.4");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
    FROM => STUDENT,
    WHERE => LIKE ( ST_NAME, ("M%") ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);

```

# UNCLASSIFIED

```

    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.23.5

--      select *
--      from GRADE ;

NEW_LINE;
PUT_LINE ("Output of Example 10.23.5");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
        FROM => GRADE ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("GRADE_COURSE GRADE_AVERAGE");
    GOT_ONE := 0;

```

# UNCLASSIFIED

```

loop
  FETCH ( CURSOR );
  INTO (V_GRADE_COURSE);
  INTO (V_GRADE_AVERAGE);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- GRADE_COURSE
  PUT (V_GRADE_COURSE, 3);
  SET_COL (14); -- GRADE_AVERAGE
  PUT (V_GRADE_AVERAGE, 3, 2, 0);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

```

--Example 10.23.6

```

--      insert into GRADE
--      select CLASS_COURSE, avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--      from CLASS
--      where CLASS_DEPT = 5
--      group by CLASS_COURSE ;

NEW_LINE;
PUT_LINE ("Output of Example 10.23.6");

```

```

INSERT INTO ( GRADE ,
  SELEC ( CLASS_COURSE & avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
  FROM => CLASS,
  WHERE => EQ ( CLASS_DEPT, 5 ),
  GROUP_BY => CLASS_COURSE ) ) ;

```

--Example 10.23.7

```

--      select *
--      from GRADE ;

NEW_LINE;
PUT_LINE ("Output of Example 10.23.7");

DECLAR ( CURSOR , CURSOR_FOR =>

```



# UNCLASSIFIED

```

        SELEC ( '*',
            FROM => GRADE ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ( "GRADE_COURSE GRADE_AVERAGE" );
    GOT_ONE := 0;

    loop
        FETCH ( CURSOR );
        INTO (V_GRADE_COURSE);
        INTO (V_GRADE_AVERAGE);
        GOT_ONE := GOT_ONE + 1;

        SET_COL (1); -- GRADE_COURSE
        PUT (V_GRADE_COURSE, 3);
        SET_COL (14); -- GRADE_AVERAGE
        PUT (V_GRADE_AVERAGE, 3, 2, 0);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ( "EXCEPTION: Not Found Error" );
    else
        null;
    end if;

    when NO_UPDATE_ERROR => PUT_LINE ( "EXCEPTION: No Update Error" );
    when UNIQUE_ERROR     => PUT_LINE ( "EXCEPTION: Unique Error" );
end;

CLOSE ( CURSOR );

--Example 10.23.8
--      insert into GRADE ( GRADE_AVERAGE )
--      select avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--      from CLASS ;

NEW_LINE;
PUT_LINE ( "Output of Example 10.23.8" );

INSERT INTO ( GRADE ( GRADE_COURSE & GRADE_AVERAGE ),
    SELEC ( TYPES.ADA_SQL.ID_COURSE' ( 999 ) &
        CONVERT_TO.TYPES.GRADE_POINT
            ( avg ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ) ),
        FROM => CLASS ) ) ;

--Example 10.23.9

```

# UNCLASSIFIED

```
--      select *
--      from GRADE ;

NEW_LINE;
PUT_LINE ("Output of Example 10.23.9");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
              FROM => GRADE) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("GRADE_COURSE GRADE_AVERAGE");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_GRADE_COURSE);
    INTO (V_GRADE_AVERAGE);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- GRADE_COURSE
    PUT (V_GRADE_COURSE, 3);
    SET_COL (14); -- GRADE_AVERAGE
    PUT (V_GRADE_AVERAGE, 3, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
                          PUT_LINE ("EXCEPTION: Not Found Error");
                          else
                            null;
                          end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.24.1

--      select *
--      from STUDENT
--      where ST_NAME = 'Mamout' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.24.1");
```

UNCLASSIFIED

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*' ,
    FROM => STUDENT,
    WHERE => EQ ( ST_NAME, "Mamout " ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ( "ST_ID  ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

# UNCLASSIFIED

```
CLOSE ( CURSOR );
```

```
--Example 10.24.2
```

```
--      update STUDENT
--      set ST_ID = 27,
--          ST_FIRST = 'Mark      ',
--          ST_ROOM = 'B101',
--          ST_MAJOR = 3
--      where ST_NAME = 'Mamout      ' ;
```

```
NEW_LINE;
```

```
PUT_LINE ("Output of Example 10.24.2");
```

```
UPDATE ( STUDENT,
SET => ST_ID <= 27
and ST_FIRST <= "Mark      "
and ST_ROOM <= "B101"
and ST_MAJOR <= 3,
WHERE => EQ ( ST_NAME, "Mamout      " ) ) ;
```

```
--Example 10.24.3
```

```
--      select *
--      from STUDENT
--      where ST_NAME = 'Mamout      ' ;
```

```
NEW_LINE;
```

```
PUT_LINE ("Output of Example 10.24.3");
```

```
DECLAR ( CURSOR , CURSOR_FOR =>
SELEC ( '*',
FROM => STUDENT,
WHERE => EQ ( ST_NAME, "Mamout      " ) ) ) ;
```

```
OPEN ( CURSOR );
```

```
begin
```

```
NEW_LINE;
```

```
PUT ("ST_ID ST_NAME      ST_FIRST ST_ROOM ST_STATE " &
"ST_MAJOR ST_YEAR");
```

```
GOT_ONE := 0;
```

```
loop
```

```
FETCH ( CURSOR );
INTO (V_ST_ID);
INTO (V_ST_NAME, V_ST_NAME_INDEX);
INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
INTO (V_ST_STATE, V_ST_STATE_INDEX);
INTO (V_ST_MAJOR);
```

# UNCLASSIFIED

```

    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.24.4

--      select *
--      from PROFESSOR ;

NEW_LINE;
PUT_LINE ("Output of Example 10.24.4");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*' ,
        FROM => PROFESSOR ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
    GOT_ONE := 0;

```

# UNCLASSIFIED

```

loop
  FETCH ( CURSOR );
  INTO ( V_PROF_ID );
  INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
  INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
  INTO ( V_PROF_DEPT );
  INTO ( V_PROF_YEARS );
  INTO ( V_PROF_SALARY );
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- PROF_ID
  PUT (V_PROF_ID, 2);
  SET_COL (10); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
  SET_COL (24); -- PROF_FIRST
  PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
  SET_COL (36); -- PROF_DEPT
  PUT (V_PROF_DEPT, 1);
  SET_COL (47); -- PROF_YEARS
  PUT (V_PROF_YEARS, 2);
  SET_COL (59); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
  NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.24.5

--      update PROFESSOR
--      set PROF_SALARY = PROF_SALARY * 1.05
--      where PROF_YEARS > 10 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.24.5");

UPDATE ( PROFESSOR,
  SET => PROF_SALARY <= PROF_SALARY * 1.05,
  WHERE => PROF_YEARS > 10 ) ;

--Example 10.24.6

```

UNCLASSIFIED

```
--      select *
--      from PROFESSOR
--      where PROF_YEARS > 10 ;

NEW_LINE;
PUT_LINE ("Output of Example 10.24.6");

DECLAR ( CURSOR , CURSOR_FOR =>
        SELEC ( '*',
        FROM => PROFESSOR,
        WHERE => PROF_YEARS > 10 ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );
    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
```

# UNCLASSIFIED

```

when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

```

```

CLOSE ( CURSOR );

```

--Example 10.24.7

```

--      update PROFESSOR
--      set PROF_SALARY =
--          ( select ( avg ( PROF_SALARY ) * 1.05 )
--            from PROFESSOR
--            where PROF_YEARS < 5 )
--      where PROF_NAME = 'Steinbacner ' ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.24.7");

```

```

UPDATE ( PROFESSOR ,
SET => PROF_SALARY <= PROF_SALARY * 1.05,
WHERE => EQ ( PROF_NAME, "Steinbacner " ) ) ;

```

--Example 10.24.8

```

--      select *
--      from PROFESSOR
--      where PROF_NAME = 'Steinbacner ' ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.24.8");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
SELEC ( '*',
FROM => PROFESSOR,
WHERE => EQ ( PROF_NAME, "Steinbacner " ) ) ) ;

```

```

OPEN ( CURSOR );

```

begin

```

NEW_LINE;
PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
"PROF_YEARS  PROF_SALARY");
GOT_ONE := 0;

```

loop

```

    FETCH ( CURSOR );
    INTO ( V_PROF_ID );
    INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
    INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
    INTO ( V_PROF_DEPT );
    INTO ( V_PROF_YEARS );

```



UNCLASSIFIED

```

    INTO ( V_PROF_SALARY );
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- PROF_ID
    PUT (V_PROF_ID, 2);
    SET_COL (10); -- PROF_NAME
    PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
    SET_COL (24); -- PROF_FIRST
    PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
    SET_COL (36); -- PROF_DEPT
    PUT (V_PROF_DEPT, 1);
    SET_COL (47); -- PROF_YEARS
    PUT (V_PROF_YEARS, 2);
    SET_COL (59); -- PROF_SALARY
    PUT (V_PROF_SALARY, 5, 2, 0);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR     => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.24.10

--      select *
--      from PROFESSOR ;

NEW_LINE;
PUT_LINE ("Output of Example 10.24.10");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
        FROM => PROFESSOR ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("PROF_ID  PROF_NAME      PROF_FIRST  PROF_DEPT  " &
        "PROF_YEARS  PROF_SALARY");
    GOT_ONE := 0;

    loop

```

UNCLASSIFIED

```

FETCH ( CURSOR );
INTO ( V_PROF_ID );
INTO ( V_PROF_NAME, V_PROF_NAME_INDEX );
INTO ( V_PROF_FIRST, V_PROF_FIRST_INDEX );
INTO ( V_PROF_DEPT );
INTO ( V_PROF_YEARS );
INTO ( V_PROF_SALARY );
GOT_ONE := GOT_ONE + 1;

SET_COL (1); -- PROF_ID
  PUT (V_PROF_ID, 2);
SET_COL (10); -- PROF_NAME
  PUT (V_PROF_NAME, V_PROF_NAME_INDEX);
SET_COL (24); -- PROF_FIRST
  PUT (V_PROF_FIRST, V_PROF_FIRST_INDEX);
SET_COL (36); -- PROF_DEPT
  PUT (V_PROF_DEPT, 1);
SET_COL (47); -- PROF_YEARS
  PUT (V_PROF_YEARS, 2);
SET_COL (59); -- PROF_SALARY
  PUT (V_PROF_SALARY, 5, 2, 0);
NEW_LINE;
end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.25.1

--      select *
--      from STUDENT
--      where ST_NAME = 'Bennett' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.1");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => STUDENT,
    WHERE => EQ ( ST_NAME, "Bennett" ) ) );

OPEN ( CURSOR );

```

# UNCLASSIFIED

```

begin
  NEW_LINE;
  PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
       "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;
  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.25.2

--      delete STUDENT
--      where ST_NAME =      'Bennett      ' ,

```

# UNCLASSIFIED

```

NEW_LINE;
PUT_LINE ("Output of Example 10.25.2");

DELETE_FROM ( STUDENT,
  WHERE => EQ ( ST_NAME, "Bennett      " ) ) ;

--Example 10.25.3

--      select *
--      from STUDENT
--      where ST_NAME = 'Martin      ' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.3");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => STUDENT,
    WHERE => EQ ( ST_NAME, "Martin      " ) ) ) ;

OPEN ( CURSOR );

begin
NEW_LINE;
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
  "ST_MAJOR  ST_YEAR");
GOT_ONE := 0;

loop
  FETCH ( CURSOR );
  INTO (V_ST_ID);
  INTO (V_ST_NAME, V_ST_NAME_INDEX);
  INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
  INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
  INTO (V_ST_STATE, V_ST_STATE_INDEX);
  INTO (V_ST_MAJOR);
  INTO (V_ST_YEAR);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- ST_ID
  PUT (V_ST_ID, 3);
  SET_COL (8); -- ST_NAME
  PUT (V_ST_NAME, V_ST_NAME_INDEX);
  SET_COL (22); -- ST_FIRST
  PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
  SET_COL (34); -- ST_ROOM
  PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
  SET_COL (43); -- ST_STATE
  PUT (V_ST_STATE, V_ST_STATE_INDEX);
  SET_COL (53); -- ST_MAJOR
  PUT (V_ST_MAJOR, 1);

```

UNCLASSIFIED

```

        SET_COL (63);  -- ST_YEAR
        PUT (V_ST_YEAR);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.25.4

--      delete STUDENT
--      where ST_NAME =      'Martin      '
--      and ST_FIRST =      'Edward      ' ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.4");

DELETE_FROM ( STUDENT ,
    WHERE => EQ ( ST_NAME, "Martin      " )
    AND EQ ( ST_FIRST, "Edward      " ) ) ;

--Example 10.25.5

--      select *
--      from STUDENT
--      where ST_NAME = 'Bennett      '
--      or ST_NAME = 'Martin      ' ;

NEW_LINE;
PUT_LINE ("Output of Example 10 25.5");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*',
        FROM => STUDENT,
        WHERE => EQ ( ST_NAME, "Bennett      " )
        OR EQ ( ST_NAME, "Martin      " ) ) ) ;

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT ("ST_ID  ST_NAME      ST_FIRST      ST_ROOM  ST_STATE  " &

```

# UNCLASSIFIED

```

        "ST_MAJOR ST_YEAR");
GOT_ONE := 0;

loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
    SET_COL (43); -- ST_STATE
    PUT (V_ST_STATE, V_ST_STATE_INDEX);
    SET_COL (53); -- ST_MAJOR
    PUT (V_ST_MAJOR, 1);
    SET_COL (63); -- ST_YEAR
    PUT (V_ST_YEAR);
    NEW_LINE;
end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.25.6

--      select CLASS_STUDENT, CLASS_COURSE, ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2
--      from CLASS
--      where CLASS_STUDENT =
--            ( select CLASS_STUDENT
--              from CLASS
--              where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =

```

# UNCLASSIFIED

```
--          ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--          from CLASS ) );

NEW_LINE;
PUT_LINE ("Output of Example 10.25.6");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( CLASS_STUDENT & CLASS_COURSE &
    ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
  FROM => CLASS,
  WHERE => EQ ( CLASS_STUDENT,
    SELEC ( CLASS_STUDENT,
      FROM => CLASS,
      WHERE => EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
        SELEC ( min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
          FROM => CLASS ) ) ) ) );

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT_LINE ("CLASS_STUDENT  CLASS_COURSE  CLASS_SEM_1 + CLASS_SEM_2 / 2.00");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_CLASS_STUDENT);
    INTO (V_CLASS_COURSE);
    INTO (AVG_SEM_1);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- CLASS_STUDENT
    PUT (V_CLASS_STUDENT, 3);
    SET_COL (16); -- CLASS_COURSE
    PUT (V_CLASS_COURSE, 3);
    SET_COL (31); -- AVG_SEM_1
    PUT (AVG_SEM_1, 3, 2, 0);
    NEW_LINE;
  end loop;

exception
  when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
  else
    null;
  end if;

  when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
  when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );
```

# UNCLASSIFIED

--Example 10.25.7

```
--      select *
--      from STUDENT
--      where ST_ID =
--      ( select CLASS_STUDENT
--      from CLASS
--      where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--      ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--      from CLASS ) ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.7");

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => STUDENT,
    WHERE => EQ ( ST_ID,
      SELEC ( CLASS_STUDENT,
        FROM => CLASS,
        WHERE => EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
          SELEC ( min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
            FROM => CLASS ) ) ) ) ) ;

OPEN ( CURSOR );

begin
  NEW_LINE;
  PUT ("ST_ID ST_NAME          ST_FIRST  ST_ROOM  ST_STATE  " &
    "ST_MAJOR  ST_YEAR");
  GOT_ONE := 0;

  loop
    FETCH ( CURSOR );
    INTO (V_ST_ID);
    INTO (V_ST_NAME, V_ST_NAME_INDEX);
    INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
    INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
    INTO (V_ST_STATE, V_ST_STATE_INDEX);
    INTO (V_ST_MAJOR);
    INTO (V_ST_YEAR);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- ST_ID
    PUT (V_ST_ID, 3);
    SET_COL (8); -- ST_NAME
    PUT (V_ST_NAME, V_ST_NAME_INDEX);
    SET_COL (22); -- ST_FIRST
    PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
    SET_COL (34); -- ST_ROOM
    PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
```



# UNCLASSIFIED

```

SET_COL (43); -- ST_STATE
PUT (V_ST_STATE, V_ST_STATE_INDEX);
SET_COL (53); -- ST_MAJOR
PUT (V_ST_MAJOR, 1);
SET_COL (63); -- ST_YEAR
PUT (V_ST_YEAR);
NEW_LINE;
end loop;

exception
when NOT_FOUND_ERROR => if GOT_ONE = 0 then
    PUT_LINE ("EXCEPTION: Not Found Error");
else
    null;
end if;
when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.25.8

--      delete STUDENT
--      where ST_ID =
--      ( select CLASS_STUDENT
--        from CLASS
--        where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--              ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--                from CLASS ) ) ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.8");

DELETE_FROM ( STUDENT,
WHERE => EQ ( ST_ID,
SELEC ( CLASS_STUDENT,
FROM => CLASS,
WHERE => EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
SELEC ( min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
FROM => CLASS ) ) ) ) ;

--Example 10.25.9

--      delete CLASS
--      where CLASS_STUDENT =
--      ( select CLASS_STUDENT
--        from CLASS
--        where ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 =
--              ( select min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2 )
--                from CLASS ) ) ;

```

# UNCLASSIFIED

```

NEW_LINE;
PUT_LINE ("Output of Example 10.25.9");

DELETE_FROM ( CLASS,
  WHERE => EQ ( CLASS_STUDENT,
    SELEC ( CLASS_STUDENT,
      FROM => CLASS,
        WHERE => EQ ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00,
          SELEC ( min ( ( CLASS_SEM_1 + CLASS_SEM_2 ) / 2.00 ),
            FROM => CLASS ) ) ) ) );

```

--Example 10.25.10

```

--      select *
--      from STUDENT ;

```

```

NEW_LINE;
PUT_LINE ("Output of Example 10.25.10");

```

```

DECLAR ( CURSOR , CURSOR_FOR =>
  SELEC ( '*',
    FROM => STUDENT ) );

```

```

OPEN ( CURSOR );

```

begin

```

NEW_LINE;
PUT ("ST_ID  ST_NAME      ST_FIRST  ST_ROOM  ST_STATE  " &
  "ST_MAJOR  ST_YEAR");
GOT_ONE := 0;

```

loop

```

  FETCH ( CURSOR );
  INTO (V_ST_ID);
  INTO (V_ST_NAME, V_ST_NAME_INDEX);
  INTO (V_ST_FIRST, V_ST_FIRST_INDEX);
  INTO (V_ST_ROOM, V_ST_ROOM_INDEX);
  INTO (V_ST_STATE, V_ST_STATE_INDEX);
  INTO (V_ST_MAJOR);
  INTO (V_ST_YEAR);
  GOT_ONE := GOT_ONE + 1;

  SET_COL (1); -- ST_ID
  PUT (V_ST_ID, 3);
  SET_COL (8); -- ST_NAME
  PUT (V_ST_NAME, V_ST_NAME_INDEX);
  SET_COL (22); -- ST_FIRST
  PUT (V_ST_FIRST, V_ST_FIRST_INDEX);
  SET_COL (34); -- ST_ROOM
  PUT (V_ST_ROOM, V_ST_ROOM_INDEX);
  SET_COL (43); -- ST_STATE

```

# UNCLASSIFIED

```

        PUT (V_ST_STATE, V_ST_STATE_INDEX);
        SET_COL (53); -- ST_MAJOR
        PUT (V_ST_MAJOR, 1);
        SET_COL (63); -- ST_YEAR
        PUT (V_ST_YEAR);
        NEW_LINE;
    end loop;

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR      => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.25.11

--      delete GRADE ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.11");

begin
    DELETE_FROM ( GRADE ) ;
exception
    when NO_UPDATE_ERROR => PUT_LINE ("NO_UPDATE_ERROR: deleting grade");
end;

--Example 10.25.12

--      select *
--      from GRADE ;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.12");

DECLAR ( CURSOR , CURSOR_FOR =>
    SELEC ( '*' ,
        FROM => GRADE ) );

OPEN ( CURSOR );

begin
    NEW_LINE;
    PUT_LINE ("GRADE_COURSE GRADE_AVERAGE");
    GOT_ONE := 0;

```

UNCLASSIFIED

```

loop
    FETCH ( CURSOR );
    INTO (V_GRADE_COURSE);
    INTO (V_GRADE_AVERAGE);
    GOT_ONE := GOT_ONE + 1;

    SET_COL (1); -- GRADE_COURSE
    PUT (V_GRADE_COURSE, 3);
    SET_COL (14); -- GRADE_AVERAGE
    PUT (V_GRADE_AVERAGE, 3, 2, 0);
    NEW_LINE;
end loop.

exception
    when NOT_FOUND_ERROR => if GOT_ONE = 0 then
        PUT_LINE ("EXCEPTION: Not Found Error");
    else
        null;
    end if;
    when NO_UPDATE_ERROR => PUT_LINE ("EXCEPTION: No Update Error");
    when UNIQUE_ERROR    => PUT_LINE ("EXCEPTION: Unique Error");
end;

CLOSE ( CURSOR );

--Example 10.25.13

--      delete DEPARTMENT;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.13");

begin
    DELETE_FROM ( DEPARTMENT ) ;
exception
    when NO_UPDATE_ERROR => PUT_LINE ("NO_UPDATE_ERROR: deleting department");
end;

--Example 10.25.14

--      delete PROFESSOR;

NEW_LINE;
PUT_LINE ("Output of Example 10.25.14");

begin
    DELETE_FROM ( PROFESSOR ) ;
exception
    when NO_UPDATE_ERROR => PUT_LINE ("NO_UPDATE_ERROR: deleting professor");
end;

```

UNCLASSIFIED

--Example 10.25.15

-- delete COURSE;

NEW\_LINE;  
PUT\_LINE ("Output of Example 10.25.15");

begin  
DELETE\_FROM ( COURSE ) ;  
exception  
when NO\_UPDATE\_ERROR => PUT\_LINE ("NO\_UPDATE\_ERROR: deleting course");  
end;

--Example 10.25.16

-- delete STUDENT;

NEW\_LINE;  
PUT\_LINE ("Output of Example 10.25.16");

begin  
DELETE\_FROM ( STUDENT ) ;  
exception  
when NO\_UPDATE\_ERROR => PUT\_LINE ("NO\_UPDATE\_ERROR: deleting student");  
end;

--Example 10.25.17

-- delete CLASS;

NEW\_LINE;  
PUT\_LINE ("Output of Example 10.25.17");

begin  
DELETE\_FROM ( CLASS ) ;  
exception  
when NO\_UPDATE\_ERROR => PUT\_LINE ("NO\_UPDATE\_ERROR: deleting class");  
end;

--Example 10.25.18

-- delete SALARY;

NEW\_LINE;  
PUT\_LINE ("Output of Example 10.25.18");

begin  
DELETE\_FROM ( SALARY ) ;  
exception  
when NO\_UPDATE\_ERROR => PUT\_LINE ("NO\_UPDATE\_ERROR: deleting salary");  
end;

## UNCLASSIFIED

```
EXIT_DATABASE;  
  
end EXAMPLES;
```

### 11.7 Output From The Sample Program

#### Output of Example 10.1.1.1

```
DEPT_ID  DEPT_DESC  
EXCEPTION: Not Found Error
```

#### Output of Example 10.1.1.2

```
EXCEPTION: Not Found Error
```

#### Output of Example 10.1.2.1

```
DEPT_DESC  
EXCEPTION: Not Found Error
```

#### Output of Example 10.1.2.2

```
EXCEPTION: Not Found Error
```

#### Output of Example 10.2.1

#### Output of Example 10.2.2

```
DEPT_ID  DEPT_DESC  
1      History
```

#### Output of Example 10.2.3

```
DEPT_DESC  
History
```

#### Output of Example 10.2.4

#### Output of Example 10.2.5

#### Output of Example 10.2.6

#### Output of Example 10.2.7

#### Output of Example 10.2.8

```
DEPT_ID  DEPT_DESC  
1      History  
2      Math  
3      Science  
4      Language  
5      Art
```

UNCLASSIFIED

Output of Example 10.2.9

Output of Example 10.2.10

Output of Example 10.2.11

Output of Example 10.2.12

Output of Example 10.2.13

Output of Example 10.2.14

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

Output of Example 10.2.15

Output of Example 10.2.16

Output of Example 10.2.17

Output of Example 10.2.18

Output of Example 10.2.19

Output of Example 10.2.20

Output of Example 10.2.21

Output of Example 10.2.22

Output of Example 10.2.23

Output of Example 10.2.24

Output of Example 10.2.25

Output of Example 10.2.26

Output of Example 10.2.27

Output of Example 10.2.28

Output of Example 10.2.29

Output of Example 10.2.30

UNCLASSIFIED

Output of Example 10.2.31

COURSE_ID	COURSE_DEPT	COURSE_DESC	COURSE_PROF	COURSE_HOURS
101	1	World History	5	TWO
102	1	Political History	5	THREE
103	1	Ancient History	5	TWO
201	2	Algebra	3	FOUR
202	2	Geometry	3	FOUR
203	2	Trigonometry	3	FIVE
204	2	Calculus	3	FOUR
301	3	Chemistry	1	THREE
302	3	Physics	1	FIVE
303	3	Biology	1	FOUR
401	4	French	2	TWO
402	4	Spanish	5	TWO
403	4	Russian	2	FOUR
501	5	Sculpture	4	ONE
502	5	Music	4	ONE
503	5	Dance	5	TWO

Output of Example 10.2.32

Output of Example 10.2.33

Output of Example 10.2.34

Output of Example 10.2.35

Output of Example 10.2.36

Output of Example 10.2.37

Output of Example 10.2.38

Output of Example 10.2.39

Output of Example 10.2.40

Output of Example 10.2.41

Output of Example 10.2.42

Output of Example 10.2.43

Output of Example 10.2.44

Output of Example 10.2.45

Output of Example 10.2.46

Output of Example 10.2.47



# UNCLASSIFIED

Output of Example 10.2.48

Output of Example 10.2.49

Output of Example 10.2.50

Output of Example 10.2.51

Output of Example 10.2.52

Output of Example 10.2.53

Output of Example 10.2.54

Output of Example 10.2.55

Output of Example 10.2.56

Output of Example 10.2.57

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR
2	McGinn	Gregory	A102	MD	1	THREE
3	Lewis	Molly	A103	PA	4	TWO
4	Waxler	Dennis	A104	NC	2	TWO
5	McNamara	Howard	A201	VA	5	ONE
6	Hess	Fay	A202	DC	3	THREE
7	Guiffre	Jennifer	A203	MD	4	ONE
8	Hagan	Carl	A204	PA	5	FOUR
9	Bearman	Rose	A301	VA	2	ONE
10	Thompson	Paul	A302	NC	1	THREE
11	Bennett	Nellie	A303	PA	4	THREE
12	Schmidt	John	A304	SC	5	TWO
13	Gevarter	Susan	B101	NY	5	FOUR
14	Sherman	Donald	B102	VA	3	THREE
15	Gorham	Milton	B103	WV	2	TWO
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR
18	Ratliff	Ann	B202	NY	3	ONE
19	Phung	Kim	B203	SC	2	TWO
20	McMurray	Eric	B204	VA	2	ONE
21	O'Leary	Peggy	C101	PA	3	FOUR
22	Martin	Charlotte	C102	DC	1	TWO
23	O'Day	Hilda	C103	NC	4	ONE
24	Martin	Edward	C104	MD	5	THREE
25	Chateaufneuf	Chelsea	C105	VA	1	THREE

Output of Example 10.2.58

Output of Example 10.2.59

**UNCLASSIFIED**

**Output of Example 10.2.60**

**Output of Example 10.2.61**

**Output of Example 10.2.62**

**Output of Example 10.2.63**

**Output of Example 10.2.64**

**Output of Example 10.2.65**

**Output of Example 10.2.66**

**Output of Example 10.2.67**

**Output of Example 10.2.68**

**Output of Example 10.2.69**

**Output of Example 10.2.70**

**Output of Example 10.2.71**

**Output of Example 10.2.72**

**Output of Example 10.2.73**

**Output of Example 10.2.74**

**Output of Example 10.2.75**

**Output of Example 10.2.76**

**Output of Example 10.2.77**

**Output of Example 10.2.78**

**Output of Example 10.2.79**

**Output of Example 10.2.80**

**Output of Example 10.2.81**

**Output of Example 10.2.82**

**Output of Example 10.2.83**

**Output of Example 10.2.84**

**Output of Example 10.2.85**

# UNCLASSIFIED

Output of Example 10.2.86

Output of Example 10.2.87

Output of Example 10.2.88

Output of Example 10.2.89

Output of Example 10.2.90

Output of Example 10.2.91

Output of Example 10.2.92

Output of Example 10.2.93

Output of Example 10.2.94

Output of Example 10.2.95

CLASS_STUDENT	CLASS_DEPT	CLASS_COURSE	CLASS_SEM_1	CLASS_SEM_2	CLASS_GRADE
1	3	302	89.49	51.91	0.00
1	3	303	77.61	88.84	0.00
2	1	103	54.38	84.77	0.00
3	4	403	92.92	97.48	0.00
4	2	204	71.17	70.55	0.00
5	5	503	88.83	81.12	0.00
6	3	301	66.26	94.60	0.00
6	4	402	100.00	100.00	0.00
7	4	401	100.00	100.00	0.00
7	4	402	100.00	100.00	0.00
7	4	403	100.00	100.00	0.00
7	5	503	100.00	100.00	0.00
8	5	502	69.68	56.92	0.00
9	2	204	55.53	89.81	0.00
10	1	102	93.72	99.55	0.00
11	4	401	81.99	76.29	0.00
12	5	501	75.81	83.03	0.00
13	5	502	67.36	80.15	0.00
14	3	302	92.27	82.47	0.00
15	2	202	89.75	95.74	0.00
16	1	101	85.64	78.26	0.00
16	1	101	94.59	91.52	0.00
16	2	204	83.40	94.88	0.00
16	3	302	82.14	87.11	0.00
16	4	403	89.92	97.40	0.00
16	5	501	76.86	95.72	0.00
17	4	401	94.71	63.36	0.00
18	5	503	92.69	71.69	0.00
19	2	201	81.31	95.95	0.00
20	2	204	88.28	79.01	0.00

# UNCLASSIFIED

21	3	303	71.16	74.14	0.00
22	1	102	58.97	86.58	0.00
22	2	201	81.75	92.97	0.00
22	5	503	74.49	98.30	0.00
23	4	402	96.33	81.53	0.00
24	5	503	97.14	85.72	0.00
25	1	101	83.58	89.16	0.00

Output of Example 10.2.96

Output of Example 10.2.97

Output of Example 10.2.98

Output of Example 10.2.99

Output of Example 10.2.100

Output of Example 10.2.101

Output of Example 10.2.102

Output of Example 10.2.103

Output of Example 10.2.104

Output of Example 10.2.105

SAL_YEAR	SAL_END	SAL_MIN	SAL_MAX	SAL_RAISE
1	1	20000.00	29999.00	0.010
2	2	30000.00	34999.00	0.075
3	3	35000.00	39999.00	0.050
4	4	40000.00	44999.00	0.035
5	5	45000.00	49999.00	0.025
6	10	50000.00	51999.00	0.020
11	15	52000.00	53999.00	0.020
16	20	54000.00	55999.00	0.020
21	99	56000.00	60000.00	0.020

Output of Example 10.3.1

ST_FIRST	ST_NAME	ST_ROOM	ST_YEAR
William	Horrigan	A101	FOUR
Gregory	McGinn	A102	THREE
Molly	Lewis	A103	TWO
Dennis	Waxler	A104	TWO
Howard	McNamara	A201	ONE
Fay	Hess	A202	THREE
Jennifer	Guiffre	A203	ONE
Carl	Hagan	A204	FOUR
Rose	Bearman	A301	ONE

# UNCLASSIFIED

Paul	Thompson	A302	THREE
Nellie	Bennett	A303	THREE
John	Schmidt	A304	TWO
Susan	Gevarter	B101	FOUR
Donald	Sherman	B102	THREE
Milton	Gorham	B103	TWO
Alvin	Williams	B104	ONE
Dorothy	Woodliff	B201	FOUR
Ann	Ratliff	B202	ONE
Kim	Phung	B203	TWO
Eric	McMurray	B204	ONE
Peggy	O'Leary	C101	FOUR
Charoltte	Martin	C102	TWO
Hilda	O'Day	C103	ONE
Edward	Martin	C104	THREE
Chelsea	Chateauneuf	C105	THREE

## Output of Example 10.3.2

PROF_NAME	PROF_SALARY	PROF_YEARS
Dysart	35000.00	3
Hall	45000.00	7
Steinbacner	30000.00	1
Bailey	50000.00	15
Clements	40000.00	4

## Output of Example 10.4.1

ST\_STATE

VA  
MD  
PA  
NC  
VA  
DC  
MD  
PA  
VA  
NC  
PA  
SC  
NY  
VA  
WV  
DC  
MD  
NY  
SC  
VA  
PA  
DC

# UNCLASSIFIED

NC  
MD  
VA

## Output of Example 10.4.2

ST\_STATE  
DC  
MD  
NC  
NY  
PA  
SC  
VA  
WV

## Output of Example 10.4.3

ST_STATE	ST_YEAR
DC	ONE
DC	TWO
DC	THREE
MD	ONE
MD	THREE
MD	FOUR
NC	ONE
NC	TWO
NC	THREE
NY	ONE
NY	FOUR
PA	TWO
PA	THREE
PA	FOUR
SC	TWO
VA	ONE
VA	THREE
VA	FOUR
WV	TWO

## Output of Example 10.6.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR
5	McNamara	Howard	A201	VA	5	ONE
9	Bearman	Rose	A301	VA	2	ONE
14	Sherman	Donald	B102	VA	3	THREE
20	McMurray	Eric	B204	VA	2	ONE
25	Chateaufneuf	Chelsea	C105	VA	1	THREE

## Output of Example 10.6.2

UNCLASSIFIED

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
2	McGinn	Gregory	A102	MD	1	THREE

Output of Example 10.6.3

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
3	Steinbacner	Moris	2	1	30000.00

Output of Example 10.6.4

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

Output of Example 10.6.5

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

Output of Example 10.6.6

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
2	Hall	Elizabeth	4	7	45000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

Output of Example 10.6.7

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
3	Steinbacner	Moris	2	1	30000.00

Output of Example 10.6.8

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
3	Steinbacner	Moris	2	1	30000.00

Output of Example 10.6.9

CLASS_STUDENT	CLASS_DEPT	CLASS_COURSE	CLASS_SEM_1	CLASS_SEM_2	CLASS_GRADE
1	3	302	89.49	51.91	0.00
4	2	204	71.17	70.55	0.00
5	5	503	88.83	81.12	0.00
8	5	502	69.68	56.92	0.00
11	4	401	81.99	76.29	0.00

# UNCLASSIFIED

14	3	302	92.27	82.47	0.00
16	1	101	85.64	78.26	0.00
16	1	101	94.59	91.52	0.00
17	4	401	94.71	63.36	0.00
18	5	503	92.69	71.69	0.00
20	2	204	88.28	79.01	0.00
23	4	402	96.33	81.53	0.00
24	5	503	97.14	85.72	0.00

## Output of Example 10.7.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
5	McNamara	Howard	A201	VA	5	ONE
9	Bearman	Rose	A301	VA	2	ONE
20	McMurray	Eric	B204	VA	2	ONE

## Output of Example 10.7.2

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
4	Waxler	Dennis	A104	NC	2	TWO
10	Thompson	Paul	A302	NC	1	THREE
12	Schmidt	John	A304	SC	5	TWO
19	Phung	Kim	B203	SC	2	TWO
23	O'Day	Hilda	C103	NC	4	ONE

## Output of Example 10.7.3

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
4	Waxler	Dennis	A104	NC	2	TWO
12	Schmidt	John	A304	SC	5	TWO
19	Phung	Kim	B203	SC	2	TWO

## Output of Example 10.7.4

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
5	Clements	Carol	1	4	40000.00

## Output of Example 10.7.5

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR
2	McGinn	Gregory	A102	MD	1	THREE
4	Waxler	Dennis	A104	NC	2	TWO
5	McNamara	Howard	A201	VA	5	ONE
6	Hess	Fay	A202	DC	3	THREE
8	Hagan	Carl	A204	PA	5	FOUR
9	Bearman	Rose	A301	VA	2	ONE
12	Schmidt	John	A304	SC	5	TWO
13	Gevarter	Susan	B101	NY	5	FOUR
16	Williams	Alvin	B104	DC	1	ONE



# UNCLASSIFIED

17	Woodliff	Dorothy	B201	MD	4	FOUR
19	Phung	Kim	B203	SC	2	TWO
20	McMurray	Eric	B204	VA	2	ONE
21	O'Leary	Peggy	C101	PA	3	FOUR
22	Martin	Charoltte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE

## Output of Example 10.8.1

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
5	Clements	Carol	1	4	40000.00

## Output of Example 10.8.2

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
5	Clements	Carol	1	4	40000.00

## Output of Example 10.9.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR
2	McGinn	Gregory	A102	MD	1	THREE
5	McNamara	Howard	A201	VA	5	ONE
6	Hess	Fay	A202	DC	3	THREE
7	Guiffre	Jennifer	A203	MD	4	ONE
9	Bearman	Rose	A301	VA	2	ONE
14	Sherman	Donald	B102	VA	3	THREE
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR
20	McMurray	Eric	B204	VA	2	ONE
22	Martin	Charoltte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE
25	Chateaufneuf	Chelsea	C105	VA	1	THREE

## Output of Example 10.9.2

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
1	Horrigan	William	A101	VA	3	FOUR
2	McGinn	Gregory	A102	MD	1	THREE
5	McNamara	Howard	A201	VA	5	ONE
6	Hess	Fay	A202	DC	3	THREE
7	Guiffre	Jennifer	A203	MD	4	ONE
9	Bearman	Rose	A301	VA	2	ONE
14	Sherman	Donald	B102	VA	3	THREE
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR
20	McMurray	Eric	B204	VA	2	ONE

UNCLASSIFIED

22	Martin	Charlotte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE
25	Chateauneuf	Chelsea	C105	VA	1	THREE

**Output of Example 10.11.1**

ST\_NAME  
Schmidt  
Sherman

**Output of Example 10.11.2**

ST_NAME	ST_ROOM
Horrigan	A101
McGinn	A102
Lewis	A103
Waxler	A104
McNamara	A201
Hess	A202
Guiffre	A203
Hagan	A204
Bearman	A301
Thompson	A302
Bennett	A303
Schmidt	A304

**Output of Example 10.11.3**

ST_NAME	ST_ROOM
Horrigan	A101
McGinn	A102
Lewis	A103
Waxler	A104
McNamara	A201
Hess	A202
Guiffre	A203
Hagan	A204
Bearman	A301
Thompson	A302
Bennett	A303
Schmidt	A304

**Output of Example 10.11.4**

ST_NAME	ST_ROOM
Horrigan	A101
Gevarter	B101
O'Leary	C101

**Output of Example 10.12.1**

# UNCLASSIFIED

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
2	McGinn	Gregory	A102	MD	1	THREE
3	Lewis	Molly	A103	PA	4	TWO
4	Waxler	Dennis	A104	NC	2	TWO
6	Hess	Fay	A202	DC	3	THREE
7	Guiffre	Jennifer	A203	MD	4	ONE
8	Hagan	Carl	A204	PA	5	FOUR
10	Thompson	Paul	A302	NC	1	THREE
11	Bennett	Nellie	A303	PA	4	THREE
12	Schmidt	John	A304	SC	5	TWO
13	Gevarter	Susan	B101	NY	5	FOUR
15	Gorham	Milton	B103	WV	2	TWO
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR
18	Ratliff	Ann	B202	NY	5	ONE
19	Phung	Kim	B203	SC	2	TWO
21	O'Leary	Peggy	C101	PA	3	FOUR
22	Martin	Charoltte	C102	DC	1	TWO
23	O'Day	Hilda	C103	NC	4	ONE
24	Martin	Edward	C104	MD	5	THREE

## Output of Example 10.12.2

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00

## Output of Example 10.12.3

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00

## Output of Example 10.12.4

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
3	Lewis	Molly	A103	PA	4	TWO
4	Waxler	Dennis	A104	NC	2	TWO
8	Hagan	Carl	A204	PA	5	FOUR
10	Thompson	Paul	A302	NC	1	THREE
11	Bennett	Nellie	A303	PA	4	THREE
12	Schmidt	John	A304	SC	5	TWO
13	Gevarter	Susan	B101	NY	5	FOUR
15	Gorham	Milton	B103	WV	2	TWO
18	Ratliff	Ann	B202	NY	5	ONE
19	Phung	Kim	B203	SC	2	TWO
21	O'Leary	Peggy	C101	PA	3	FOUR
23	O'Day	Hilda	C103	NC	4	ONE

## Output of Example 10.12.5

# UNCLASSIFIED

ST_NAME	ST_ROOM
Gevarter	B101
Sherman	B102
Gorham	B103
Williams	B104
Woodliff	B201
Ratliff	B202
Phung	B203
McMurray	B204
O'Leary	C101
Martin	C102
O'Day	C103
Martin	C104
Chateauneuf	C105

## Output of Example 10.13.1

PROF_NAME	PROF_SALARY * 1.10
Dysart	38500.00
Hall	49500.00
Steinbacner	33000.00
Bailey	55000.00
Clements	44000.00

## Output of Example 10.13.2

CLASS_STUDENT	CLASS_SEM_1 + CLASS_SEM_2 / 2
1	70.70
1	83.22
2	69.57
3	95.20
4	70.86
5	84.97
6	80.43
6	100.00
7	100.00
7	100.00
7	100.00
7	100.00
8	63.30
9	72.67
10	96.64
11	79.14
12	79.42
13	73.75
14	87.37
15	92.75
16	81.95
16	93.06
16	89.14
16	84.63

UNCLASSIFIED

16	93.66
16	86.29
17	79.04
18	82.19
19	88.63
20	83.64
21	72.65
22	72.78
22	87.36
22	85.39
23	88.93
24	91.43
25	86.37

Output of Example 10.13.3

PROF_NAME	PROF_SALARY
Dysart	35000.00
Steinbacner	30000.00

Output of Example 10.13.4

PROF_NAME	PROF_SALARY * 1.10
Hall	49500.00
Bailey	55000.00

Output of Example 10.14.1

COUNT  
25

Output of Example 10.14.2

COUNT  
12

Output of Example 10.14.3

COUNT  
13

Output of Example 10.14.5

COUNT  
6

Output of Example 10.14.7

MINIMUM SALARY	MAXIMUM SALARY
30000.00	50000.00

UNCLASSIFIED

Output of Example 10.14.8

SALARY  
200000.00

Output of Example 10.14.9

AVERAGE CLASS_SEM_1	AVERAGE CLASS_SEM_2
85.43	90.82

Output of Example 10.14.10

COUNT	SALARY	SUM	AVERAGE	MINIMUM	MAXIMUM
5		200000.00	40000.00	30000.00	50000.00

Output of Example 10.15.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
9	Bearman	Rose	A301	VA	2	ONE
11	Bennett	Nellie	A303	PA	4	THREE
25	Chateauneuf	Chelsea	C105	VA	1	THREE
13	Gevarter	Susan	B101	NY	5	FOUR
15	Gorham	Milton	B103	WV	2	TWO
7	Guiffre	Jennifer	A203	MD	4	ONE
8	Hagan	Carl	A204	PA	5	FOUR
6	Hess	Fay	A202	DC	3	THREE
1	Horrigan	William	A101	VA	3	FOUR
3	Lewis	Molly	A103	PA	4	TWO
22	Martin	Charlotte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE
2	McGinn	Gregory	A102	MD	1	THREE
20	McMurray	Eric	B204	VA	2	ONE
5	McNamara	Howard	A201	VA	5	ONE
23	O'Day	Hilda	C103	NC	4	ONE
21	O'Leary	Peggy	C101	PA	3	FOUR
19	Phung	Kim	B203	SC	2	TWO
18	Ratliff	Ann	B202	NY	5	ONE
12	Schmidt	John	A304	SC	5	TWO
14	Sherman	Donald	B102	VA	3	THREE
10	Thompson	Paul	A302	NC	1	THREE
4	Waxler	Dennis	A104	NC	2	TWO
16	Williams	Alvin	B104	DC	1	ONE
17	Woodliff	Dorothy	B201	MD	4	FOUR

Output of Example 10.15.2

PROF_NAME	PROF_SALARY
Bailey	50000.00
Hall	45000.00
Clements	40000.00
Dysart	35000.00

# UNCLASSIFIED

Steinbacner 30000.00

## Output of Example 10.15.3

ST_NAME	ST_YEAR	ST_MAJOR
Horrigan	FOUR	3
O'Leary	FOUR	3
Woodliff	FOUR	4
Hagan	FOUR	5
Gevarter	FOUR	5
McGinn	THREE	1
Chateauneuf	THREE	1
Thompson	THREE	1
Hess	THREE	3
Sherman	THREE	3
Bennett	THREE	4
Martin	THREE	5
Martin	TWO	1
Waxler	TWO	2
Gorham	TWO	2
Phung	TWO	2
Lewis	TWO	4
Schmidt	TWO	5
Williams	ONE	1
Bearman	ONE	2
McMurray	ONE	2
Guiffre	ONE	4
O'Day	ONE	4
McNamara	ONE	5
Ratliff	ONE	5

## Output of Example 10.15.4

ST_NAME	ST_YEAR	ST_MAJOR
Horrigan	FOUR	3
O'Leary	FOUR	3
Woodliff	FOUR	4
Hagan	FOUR	5
Gevarter	FOUR	5
McGinn	THREE	1
Chateauneuf	THREE	1
Thompson	THREE	1
Hess	THREE	3
Sherman	THREE	3
Bennett	THREE	4
Martin	THREE	5
Martin	TWO	1
Waxler	TWO	2
Gorham	TWO	2
Phung	TWO	2
Lewis	TWO	4

UNCLASSIFIED

Schmidt	TWO	5
Williams	ONE	1
Bearman	ONE	2
McMurray	ONE	2
Guiffre	ONE	4
O'Day	ONE	4
McNamara	ONE	5
Ratliff	ONE	5

**Output of Example 10.15.5**

CLASS_DEPT	CLASS_COURSE	CLASS_SEM_1 + CLASS_SEM_2 / 2	CLASS_STUDENT
4	401	100.00	7
4	402	100.00	6
4	402	100.00	7
4	403	100.00	7
5	503	100.00	7
1	102	96.64	10
4	403	93.66	16
2	202	92.75	15

**Output of Example 10.16.1**

CLASS_STUDENT	AVG_SEM_1	AVG_SEM_2
1	83.55	70.38
2	54.38	84.77
3	92.92	97.48
4	71.17	70.55
5	88.83	81.12
6	83.13	97.30
7	100.00	100.00
8	69.68	56.92
9	55.53	89.81
10	93.72	99.55
11	81.99	76.29
12	75.81	83.03
13	67.36	80.15
14	92.27	82.47
15	89.75	95.74
16	85.43	90.82
17	94.71	63.36
18	92.69	71.69
19	81.31	95.95
20	88.28	79.01
21	71.16	74.14
22	71.74	92.62
23	96.33	81.53
24	97.14	85.72
25	83.58	89.16

**Output of Example 10.16.2**



# UNCLASSIFIED

CLASS_STUDENT	AVG_SEM_1	AVG_SEM_2
1	83.55	70.38
6	66.26	94.60
14	92.27	82.47
16	82.14	87.11
21	71.16	74.14

## Output of Example 10.16.3

CLASS_STUDENT	CLASS_DEPT	AVG_SEM_1	AVG_SEM_2
2	1	54.38	84.77
10	1	93.72	99.55
16	1	90.11	84.89
22	1	58.97	86.58
25	1	83.58	89.16
4	2	71.17	70.55
9	2	55.53	89.81
15	2	89.75	95.74
16	2	83.40	94.88
19	2	81.31	95.95
20	2	88.28	79.01
22	2	81.75	92.97
1	3	83.55	70.38
6	3	66.26	94.60
14	3	92.27	82.47
16	3	82.14	87.11
21	3	71.16	74.14
3	4	92.92	97.48
6	4	100.00	100.00
7	4	100.00	100.00
11	4	81.99	76.29
16	4	89.92	97.40
17	4	94.71	63.36
23	4	96.33	81.53
5	5	88.83	81.12
7	5	100.00	100.00
8	5	69.68	56.92
12	5	75.81	83.03
13	5	67.36	80.15
16	5	76.86	95.72
18	5	92.69	71.69
22	5	74.49	98.30
24	5	97.14	85.72

## Output of Example 10.16.4

ST_STATE	ST_MAJOR	ST_YEAR	COUNT
DC	1	ONE	1
DC	1	TWO	1
DC	3	THREE	1
MD	1	THREE	1

# UNCLASSIFIED

MD	4	ONE	1
MD	4	FOUR	1
MD	5	THREE	1
NC	1	THREE	1
NC	2	TWO	1
NC	4	ONE	1
NY	5	ONE	1
NY	5	FOUR	1
PA	3	FOUR	1
PA	4	TWO	1
PA	4	THREE	1
PA	5	FOUR	1
SC	2	TWO	1
SC	5	TWO	1
VA	1	THREE	1
VA	2	ONE	2
VA	3	THREE	1
VA	3	FOUR	1
VA	5	ONE	1
WV	2	TWO	1

## Output of Example 10.17.1

MAX\_PROF\_SALARY  
50000.00

## Output of Example 10.17.2

PROF_FIRST	PROF_NAME	PROF_SALARY
Bruce	Bailey	50000.00

## Output of Example 10.17.3

PROF_FIRST	PROF_NAME	PROF_SALARY
Bruce	Bailey	50000.00

## Output of Example 10.17.4

PROF_ID	PROF_SALARY
1	35000.00
2	45000.00
5	40000.00

## Output of Example 10.17.5

PROF_ID	PROF_SALARY	PROF_YEARS
4	50000.00	15

## Output of Example 10.17.6

CLASS\_STUDENT CLASS\_SEM\_1 + CLASS\_SEM\_2 / 2

# UNCLASSIFIED

7	100.00
7	100.00
7	100.00
7	100.00
16	81.95
16	93.66
16	86.29
16	84.63
16	89.14
16	93.06
22	72.78
22	86.39
22	87.36

## Output of Example 10.17.10

CLASS_STUDENT	CLASS_SEM_1 + CLASS_SEM_2 / 2
7	100.00
7	100.00
7	100.00
7	100.00
16	93.66
16	93.06

## Output of Example 10.17.11

CLASS_STUDENT	CLASS_SEM_1 + CLASS_SEM_2 / 2
7	100.00
7	100.00
7	100.00
7	100.00
16	81.95
16	93.66
16	86.29
16	84.63
16	89.14
16	93.06
22	72.78
22	86.39
22	87.36

## Output of Example 10.17.12

CLASS_STUDENT	CLASS_SEM_1 + CLASS_SEM_2 / 2
7	100.00
7	100.00
7	100.00
7	100.00
16	81.95
16	93.06
16	89.14

# UNCLASSIFIED

16	84.63
16	93.66
16	86.29
22	72.78
22	87.36
22	86.39

## Output of Example 10.18.1

COURSE_DEPT	COURSE_PROF
1	15
2	12
5	13

## Output of Example 10.18.2

COURSE_DEPT	COUNT
2	4

## Output of Example 10.18.3

ST_STATE	ST_MAJOR	COUNT
DC	3	1
MD	4	2
MD	5	1
NC	4	1
PA	3	1
PA	4	2
PA	5	1
VA	3	2
VA	5	1

## Output of Example 10.18.4

CLASS_COURSE	AVG CLASS_SEM_1 + CLASS_SEM_2 / 2
201	88.00
202	92.75
401	86.06
402	96.31
403	96.29

## Output of Example 10.18.6

CLASS_COURSE	AVG SEM_1	AVG SEM_2
401	92.23	79.88
402	98.78	93.84
403	94.28	98.29
502	68.52	68.54
503	90.63	87.37

## Output of Example 10.18.7

# UNCLASSIFIED

CLASS_STUDENT	CLASS_SEM_2
3	97.48
7	100.00
16	97.40

## Output of Example 10.19.1

DEPARTMENT		PROFESSOR					
ID	DESC	ID	NAME	FIRST	DEPT	YEARS	SALARY
1	History	1	Dysart	Gregory	3	3	35000.00
2	Math	1	Dysart	Gregory	3	3	35000.00
3	Science	1	Dysart	Gregory	3	3	35000.00
4	Language	1	Dysart	Gregory	3	3	35000.00
5	Art	1	Dysart	Gregory	3	3	35000.00
1	History	2	Hall	Elizabeth	4	7	45000.00
2	Math	2	Hall	Elizabeth	4	7	45000.00
3	Science	2	Hall	Elizabeth	4	7	45000.00
4	Language	2	Hall	Elizabeth	4	7	45000.00
5	Art	2	Hall	Elizabeth	4	7	45000.00
1	History	3	Steinbacner	Moris	2	1	30000.00
2	Math	3	Steinbacner	Moris	2	1	30000.00
3	Science	3	Steinbacner	Moris	2	1	30000.00
4	Language	3	Steinbacner	Moris	2	1	30000.00
5	Art	3	Steinbacner	Moris	2	1	30000.00
1	History	4	Bailey	Bruce	5	15	50000.00
2	Math	4	Bailey	Bruce	5	15	50000.00
3	Science	4	Bailey	Bruce	5	15	50000.00
4	Language	4	Bailey	Bruce	5	15	50000.00
5	Art	4	Bailey	Bruce	5	15	50000.00
1	History	5	Clements	Carol	1	4	40000.00
2	Math	5	Clements	Carol	1	4	40000.00
3	Science	5	Clements	Carol	1	4	40000.00
4	Language	5	Clements	Carol	1	4	40000.00
5	Art	5	Clements	Carol	1	4	40000.00

## Output of Example 10.19.2

PROF_FIRST	PROF_NAME	DEPT_DESC
Carol	Clements	History
Moris	Steinbacner	Math
Gregory	Dysart	Science
Elizabeth	Hall	Language
Bruce	Bailey	Art

## Output of Example 10.19.3

DEPT_DESC	COURSE_DESC	PROF_FIRST	PROF_NAME
Science	Chemistry	Gregory	Dysart
Science	Biology	Gregory	Dysart
Science	Physics	Gregory	Dysart
Language	French	Elizabeth	Hall

# UNCLASSIFIED

Language	Russian	Elizabeth	Hall
Math	Algebra	Moris	Steinbacner
Math	Calculus	Moris	Steinbacner
Math	Trigonometry	Moris	Steinbacner
Math	Geometry	Moris	Steinbacner
Art	Sculpture	Bruce	Bailey
Art	Music	Bruce	Bailey
History	World History	Carol	Clements
Art	Dance	Carol	Clements
History	Political History	Carol	Clements
Language	Spanish	Carol	Clements
History	Ancient History	Carol	Clements

## Output of Example 10.19.4

DEPT_DESC	COURSE_DESC	PROF_FIRST	PROF_NAME
History	World History	Carol	Clements
History	Political History	Carol	Clements
History	Ancient History	Carol	Clements
Math	Algebra	Moris	Steinbacner
Math	Geometry	Moris	Steinbacner
Math	Trigonometry	Moris	Steinbacner
Math	Calculus	Moris	Steinbacner
Science	Chemistry	Gregory	Dysart
Science	Physics	Gregory	Dysart
Science	Biology	Gregory	Dysart
Language	French	Elizabeth	Hall
Language	Spanish	Carol	Clements
Language	Russian	Elizabeth	Hall
Art	Sculpture	Bruce	Bailey
Art	Music	Bruce	Bailey
Art	Dance	Carol	Clements

## Output of Example 10.19.5

DEPT_DESC	COURSE_DESC	PROF_FIRST	PROF_NAME
Art	Dance	Carol	Clements
Art	Music	Bruce	Bailey
Art	Sculpture	Bruce	Bailey
History	Ancient History	Carol	Clements
History	Political History	Carol	Clements
History	World History	Carol	Clements
Language	French	Elizabeth	Hall
Language	Russian	Elizabeth	Hall
Language	Spanish	Carol	Clements
Math	Algebra	Moris	Steinbacner
Math	Calculus	Moris	Steinbacner
Math	Geometry	Moris	Steinbacner
Math	Trigonometry	Moris	Steinbacner
Science	Biology	Gregory	Dysart
Science	Chemistry	Gregory	Dysart

# UNCLASSIFIED

Science      Physics                      Gregory      Dysart

## Output of Example 10.19.6

DEPT_DESC	COURSE_DESC	PROF_NAME	ST_NAME
Art	Dance	Clements	Guiffre
History	Ancient History	Clements	McGinn
Language	French	Hall	Guiffre
Language	Russian	Hall	Guiffre
Language	Spanish	Clements	Guiffre
Language	Spanish	Clements	Hess
Science	Biology	Dysart	Horrigan
Science	Chemistry	Dysart	Hess
Science	Physics	Dysart	Horrigan

## Output of Example 10.19.7

PROF_NAME	PROF_YEARS	SAL_YEAR	SAL_END	PROF_SALARY	SAL_MIN	SAL_MAX
Steinbacner	1	1	1	30000.00	20000.00	29999.00
Dysart	3	3	3	35000.00	35000.00	39999.00
Clements	4	4	4	40000.00	40000.00	44999.00
Hall	7	6	10	45000.00	50000.00	51999.00
Bailey	15	11	15	50000.00	52000.00	53999.00

## Output of Example 10.19.8

PROF_NAME	PROF_SALARY	PROF_YEARS	SAL_YEAR	SAL_END	SAL_MIN	SAL_MAX
Steinbacner	30000.00	1	2	2	30000.00	34999.00
Dysart	35000.00	3	3	3	35000.00	39999.00
Clements	40000.00	4	4	4	40000.00	44999.00
Hall	45000.00	7	5	5	45000.00	49999.00
Bailey	50000.00	15	6	10	50000.00	51999.00

## Output of Example 10.20.1

DEPT_DESC	COURSE_DESC	PROF_NAME	ST_NAME
Language	French	Hall	Guiffre
Language	Russian	Hall	Guiffre
Art	Dance	Clements	Guiffre
Language	Spanish	Clements	Guiffre
History	World History	Clements	Williams
Science	Physics	Dysart	Williams
Language	Russian	Hall	Williams
Art	Sculpture	Bailey	Williams
History	World History	Clements	Williams
Math	Calculus	Steinbacner	Williams

## Output of Example 10.20.2

DEPT_DESC	COURSE_DESC	PROF_NAME	ST_NAME
Language	French	Hall	Guiffre

# UNCLASSIFIED

Language	Russian	Hall	Guiffre
Art	Dance	Clements	Guiffre
Language	Spanish	Clements	Guiffre
History	World History	Clements	Williams
Science	Physics	Dysart	Williams
Language	Russian	Hall	Williams
Art	Sculpture	Bailey	Williams
History	World History	Clements	Williams
Math	Calculus	Steinbacner	Williams

## Output of Example 10.21.1

PROF_NAME	PROF_SALARY
Hall	45000.00
Bailey	50000.00

## Output of Example 10.21.2

PROF_FIRST	PROF_NAME	PROF_SALARY	DEPT_DESC	COURSE_DESC
Elizabeth	Hall	45000.00	Language	French
Elizabeth	Hall	45000.00	Language	Russian
Bruce	Bailey	50000.00	Art	Sculpture
Bruce	Bailey	50000.00	Art	Music

## Output of Example 10.22.5

AVG PROF_SALARY
33571.43

## Output of Example 10.22.6

PROF_SALARY	PROF_ID	COURSE_HOURS	COURSE_ID
35000.00	1	FIVE	302
35000.00	1	FOUR	303
45000.00	2	FOUR	403
30000.00	3	FOUR	201
30000.00	3	FIVE	203
30000.00	3	FOUR	204
30000.00	3	FOUR	202

## Output of Example 10.23.1

## Output of Example 10.23.2

## Output of Example 10.23.3

## Output of Example 10.23.4

## Output of Example 10.23.5

GRADE_COURSE	GRADE_AVERAGE
--------------	---------------



# UNCLASSIFIED

EXCEPTION: Not Found Error

Output of Example 10.23.6

Output of Example 10.23.7

GRADE_COURSE	GRADE_AVERAGE
501	82.86
502	68.53
503	89.00

Output of Example 10.23.8

Output of Example 10.23.9

GRADE_COURSE	GRADE_AVERAGE
501	82.86
502	68.53
503	89.00
999	85.08

Output of Example 10.24.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
99	Mamout			AK	1	ONE

Output of Example 10.24.2

Output of Example 10.24.3

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
27	Mamout	Mark	B101	AK	3	ONE

Output of Example 10.24.4

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	30000.00
4	Bailey	Bruce	5	15	50000.00
5	Clements	Carol	1	4	40000.00

Output of Example 10.24.5

Output of Example 10.24.6

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
4	Bailey	Bruce	5	15	52500.00

Output of Example 10.24.7

UNCLASSIFIED

Output of Example 10.24.8

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
3	Steinbacner	Moris	2	1	31500.00

Output of Example 10.24.10

PROF_ID	PROF_NAME	PROF_FIRST	PROF_DEPT	PROF_YEARS	PROF_SALARY
1	Dysart	Gregory	3	3	35000.00
2	Hall	Elizabeth	4	7	45000.00
3	Steinbacner	Moris	2	1	31500.00
4	Bailey	Bruce	5	15	52500.00
5	Clements	Carol	1	4	40000.00

Output of Example 10.25.1

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
11	Bennett	Nellie	A303	PA	4	THREE

Output of Example 10.25.2

Output of Example 10.25.3

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
22	Martin	Charoltte	C102	DC	1	TWO
24	Martin	Edward	C104	MD	5	THREE

Output of Example 10.25.4

Output of Example 10.25.5

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
22	Martin	Charoltte	C102	DC	1	TWO

Output of Example 10.25.6

CLASS_STUDENT	CLASS_COURSE	CLASS_SEM_1 + CLASS_SEM_2 / 2.00
8	502	63.30

Output of Example 10.25.7

ST_ID	ST_NAME	ST_FIRST	ST_ROOM	ST_STATE	ST_MAJOR	ST_YEAR
8	Hagan	Carl	A204	PA	5	FOUR

Output of Example 10.25.8

Output of Example 10.25.9

Output of Example 10.25.10

Output of Example 10.25.11

UNCLASSIFIED

**Output of Example 10.25.12**

GRADE\_COURSE GRADE\_AVERAGE  
EXCEPTION: Not Found Error

**Output of Example 10.25.13**

**Output of Example 10.25.14**

**Output of Example 10.25.15**

**Output of Example 10.25.16**

**Output of Example 10.25.17**

**Output of Example 10.25.18**

UNCLASSIFIED

Distribution List for IDA Document D-574

NAME AND ADDRESS	NUMBER OF COPIES
<b>Sponsor</b>	
CPT Stephen Myatt WIS JPMO/DXP Room 5B19, The Pentagon Washington, D.C. 20330-6600	5
<b>Other</b>	
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Mr. Bill Allen SAIC Corporation 311 Park Place Blvd Suite 360 Clearwater, FL 34619	1
Theodore F. Elbert Professor and Chairman Department of Computer Science University of West Florida Pensacola, FL 21514-5752	1
Mr. Fred Friedman P.O. Box 576 Annandale, VA 22003	1
Ms. Kerry Hilliard 7321 Franklin Rd. Annandale, VA 22003	1
Mr. Kevin Heatwole 5124 Harford Lane Burke, VA 22015	1
Mr. Chris Neilson SAIC 311 Park Place Blvd. Suite 360 Clearwater, FL 34619-3923	1
Mr. Richie Platt 428 Gabe Ct. Denton, TX 76201	1

UNCLASSIFIED

NAME AND ADDRESS	NUMBER OF COPIES
Ms. Linn Roller General Dynamics P.O. Box 748 M-2 1786 Ft. Worth, TX 76101	1
Dr. John Salasin GTE Government Systems Corp. 1700 Research Blvd. Rockville, MD 20850	1
Mr. Eugen Vasilescu 35 Chestnut St. Malverne, Long Island, NY 11565	1
<b>CSED Review Panel</b>	
Dr. Dan Alpert, Director Program in Science, Technology & Society University of Illinois Room 201 912-1/2 West Illinois Street Urbana, Illinois 61801	1
Dr. Barry W. Boehm TRW Defense Systems Group MS R2-1094 One Space Park Redondo Beach, CA 90278	1
Dr. Ruth Davis The Pymatuning Group, Inc. 2000 N. 15th Street, Suite 707 Arlington, VA 22201	1
Dr. C.E. Hutchinson, Dean Thayer School of Engineering Dartmouth College Hanover, NH 03755	1

UNCLASSIFIED

NAME AND ADDRESS	NUMBER OF COPIES
Mr. A.J. Jordano Manager, Systems & Software Engineering Headquarters Federal Systems Division 6600 Rockledge Dr. Bethesda, MD 20817	1
Mr. Robert K. Lehto Mainstay 302 Mill St. Occoquan, VA 22125	1
Dr. John M. Palms, Vice President Academic Affairs & Professor of Physics Emory University Atlanta, GA 30322	1
Mr. Oliver Selfridge 45 Percy Road Lexington, MA 02173	1
Mr. Keith Uncapher University of Southern California Olin Hall 330A University Park Los Angeles, CA 90089-1454	1
IDA	
General W.Y. Smith, HQ	1
Mr. Philip L. Major, HQ	1
Dr. Robert E. Roberts, HQ	1
Ms. Anne Douville, CSED	1
Dr. John F. Kramer, CSED	1
Mr. Terry Mayfield, CSED	1
Ms. Katydean Price, CSED	2
Mr. Bill R. Brykczynski, CSED	5
IDA Control & Distribution Vault	3